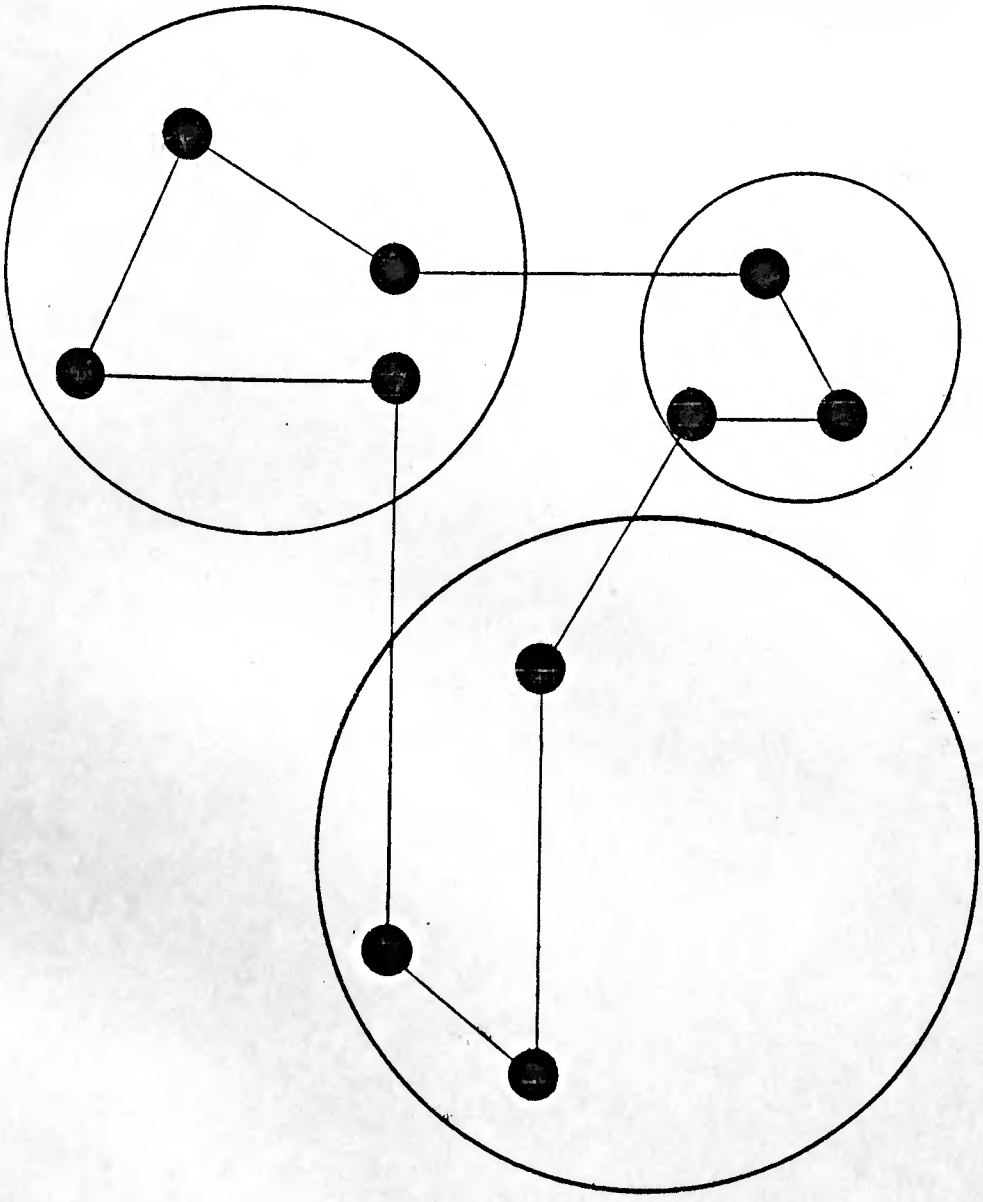
INSTITUTO SUPERIOR DE ECONOMIA
UNIVERSIDADE TÉCNICA DE LISBOA



O PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO

MÉTODOS HEURÍSTICOS E RELAXAÇÕES

LEONOR ALMEIDA LEITE SANTIAGO PINTO BRUM

JANEIRO 1988

INSTITUTO SUPERIOR DE ECONOMIA
UNIVERSIDADE TÉCNICA DE LISBOA

RESERVADO

I. S. E.
Biblioteca
I.O.
507-G. 34353
QA 164
B78
1988

O PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO

MÉTODOS HEURÍSTICOS E RELAXAÇÕES

LEONOR ALMEIDA LEITE SANTIAGO PINTO BRUM

Dissertação apresentada como requisito parcial para a obtenção do grau de mestre em Métodos Matemáticos Aplicados à Economia e Gestão de Empresas

JANEIRO 1988

Foi um privilégio ter a oportunidade de trabalhar com a Professora Doutora Teresa Chaves de Almeida, orientadora desta dissertação. A excelente relação de trabalho que proporcionou, sempre disponível para discutir o trabalho apresentando as mais valiosas sugestões, são disso prova.

Um abraço a Maria João Moura, Raúl Santiago Pinto e Leonor Vasconcelos Ferreira, que pacientemente colaboraram na execução dos pormenores gráficos necessários à apresentação deste trabalho.

Seria injusto não mencionar a solidariedade manifestada pelos colegas provando mais uma vez o óptimo ambiente de trabalho que se vive no Departamento de Matemática do ISE.

A todos, os meus mais sinceros agradecimentos.

RESUMO

O Problema do Caixeiro Viajante Classificado é uma variante do clássico Problema Caixeiro Viajante com restrições adicionais. No problema classificado as cidades estão divididas em classes exigindo-se que as cidades de cada uma das classes sejam visitadas continuamente.

Neste trabalho desenvolvem-se métodos de obtenção de majorantes e minorantes para o valor óptimo deste problema. Os majorantes foram obtidos através de soluções heurísticas e os minorantes através de relaxações Lagrangeanas no problema da determinação da árvore-1 mínima. Os diversos métodos foram testados num conjunto de 73 problemas com dimensões e estruturas de dados diversas.

ÍNDICE

1. INTRODUÇÃO

O Problema do Caixeiro Viajante Classificado, PCVC, resulta da imposição de certo tipo de restrições adicionais ao clássico Problema do Caixeiro Viajante, PCV. Este último consiste na determinação do percurso de comprimento mínimo que um caixeiro deve usar para visitar um conjunto de cidades passando por cada uma delas, uma e uma só vez e voltando à cidade de partida. No problema classificado as cidades estão divididas em classes e exige-se que o caixeiro visite o conjunto de cidades de cada classe continuamente. Na literatura encontram-se alguns exemplos de aplicações interessantes estando mesmo a primeira referência a este problema, Chisman (1975), relacionada com um exemplo concreto de recolha de produtos num armazém. No capítulo 2 apresenta-se uma formalização do PCVC, sumariam-se os métodos para a sua resolução e reunem-se alguns exemplos de aplicações.

A consideração de restrições de classe reduz substancialmente o número de soluções admissíveis, por exemplo, a divisão do conjunto de 12 cidades em 4 classes (de 3 cidades) reduz este número de 39916800 para 7776. No entanto, o PCVC parece arrastar a dificuldade de resolução inerente ao Problema do Caixeiro Viajante. Assim, como em problemas de grandes dimensões a determinação da solução óptima nem sempre é possível em tempo útil (e pode induzir a custos incomportáveis), uma forma expedita de resolver situações práticas consiste em implementar soluções aproximadas obtidas através da aplicação de métodos heurísticos. Para o problema classificado apenas Jongens e Volgenant (1985) apresentam um método deste tipo. Este consiste básicamente na determinação de uma solução para o PCV a

qual, se necessário, é sujeita a algumas alterações para forçar a verificação das restrições de classe. Parecendo, porém, mais indicada a aplicação de procedimentos destinados, logo de início, à determinar soluções para o PCVC foram concebidos alguns métodos. No capítulo 3 descrevem-se estes métodos e analisam-se os resultados de testes computacionais realizados com o objectivo de os comparar.

A resolução de problemas práticos do modo acima referido exige uma avaliação do erro cometido, isto é, uma quantificação do prejuízo incorrido pelo facto de se substituir a solução óptima por uma particular solução admissível. A informação ideal para o efeito seria a diferença entre o valor da solução aproximada e do óptimo. No entanto, o conhecimento deste último está na maior parte dos casos fora de alcance, recorrendo-se então à utilização de minorantes que proporcionam uma avaliação por excesso do referido erro. A determinação de minorantes para o valor óptimo de problemas difíceis consegue-se, com frequência, através da resolução de um problema relaxado que se obtem do primeiro através do afrouxamento, ou mesmo omissão, de um subconjunto de restrições. Os resultados da dualidade Lagrangeana em conjugação com uma escolha criteriosa do conjunto de restrições a afrouxar têm permitido a concepção de formas de obtenção de minorantes que não se afastam muito do valor óptimo do problema original. A utilidade de tais procedimentos não se confina à possibilidade acima referida de avaliação de soluções aproximadas. A grande parte dos métodos exactos para problemas difíceis pertencem à classe de métodos que habitualmente se designa por métodos de partições e avaliações sucessivas, dependendo a eficiência destes essencialmente da inclusão de uma boa forma de determinação de minorantes. No capitulo 4 deste trabalho discutem-se duas relaxações Lagrangeanas do PCVC no problema da determinação da árvore-1 míni-

ma. Neste capítulo incluem-se uma exposição dos conceitos e resultados básicos da dualidade Lagrangeana, a dedução de duas formas de obtenção de minorantes para o PCVC e ainda a análise dos resultados de experiência computacional efectuada para avaliar as qualidades das alternativas em estudo.

Por último, no capítulo 5 apresentam-se as principais conclusões e enunciam-se futuras hipóteses de trabalho.

2. FORMALIZAÇÃO E APLICAÇÕES DO PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO

O objectivo deste capítulo é apresentar o Problema do Caixeiro Viajante Classificado (como se optou por traduzir a designação inglesa "Clustered Travelling Salesman Problem"). Como este problema está intimamente relacionado com o clássico Problema do Caixeiro Viajante a primeira secção consiste numa apresentação deste último. Esta secção contem o enunciado deste problema, uma referência à dificuldade de resolução e um breve resumo das formas de abordagem mais importantes. Na segunda secção procura-se introduzir o Problema do Caixeiro Viajante Classificado. Apresenta-se um enunciado do problema e uma formalização como problema de programação linear inteira. Inclui-se ainda uma referência aos métodos de resolução. Por último, reuniram-se numa secção uma série de exemplos de algum modo relacionados com o problema em estudo.

2.1 O PROBLEMA DO CAIXEIRO VIAJANTE

O Problema do Caixeiro Viajante(*), PCV, consiste na determinação

(*) Conforme vem referido em Hoffman e Wolfe (1985), o primeiro uso do termo "Travelling Salesman Problem" em círculos matemáticos, deve ter acontecido por volta de 1931-32, acontecimento com o qual aparecem relacionados nomes como Hassler Whitney, A W Tucker e Merril Flood, sendo este último o maior responsável pela divulgação do PCV no seio da comunidade da Investigação Operacional.
Mas, curiosamente pode encontrar-se já em 1832 um primeiro esboço do enunciado do problema. Esta primeira referência aparece num livro editado na Alemanha intitulado "O Caixeiro Viajante, como deve ser e o que deve fazer para obter comissões e ser bem sucedido no seu negócio. Por um Caixeiro Viajante veterano". No último capitulo sugere-se que o caixeiro seja especialmente cuidadoso na escolha e escalonamento do seu percurso sendo o aspecto mais importante a cobertura do maior número de localidades possível sem que nenhuma seja visitada duas vezes.

do percurso de comprimento mínimo que um caixeiro deve usar para visitar um conjunto de cidades, passando por cada uma exactamente uma vez e voltando à cidade de partida (arbitrária). De forma equivalente, pode apresentar-se o PCV como sendo o da determinação do circuito Hamiltoniano de comprimento mínimo num grafo G=(N,A) onde N={i:i=1,2...,n}, o conjunto dos nodos, representa o conjunto de n cidades e A={(i,j): i∈N, j∈N e a ligação da cidade i à j existe}, o conjunto dos arcos, representa as ligações entre cidades a cada uma das quais se associa um valor numérico (c_{ij}) representando a distância da ligação (directa) da cidade i à cidade j.

Quando as ligações (i,j) e (j,i) são igualmente longas, isto é, o sentido das ligações é irrelevante o problema toma a designação de Caixeiro Viajante simétrico ou não orientado, caso contrário, denomina-se Caixeiro Viajante assimétrico ou orientado.

Este problema apesar de ser extremamente fácil de estabelecer, não requerendo grandes conhecimentos matemáticos para o efeito, tem, no entanto, resistido a todas as tentativas de encontrar um "bom" algoritmo de optimização, isto é, um método cujo tempo necessário para a sua execução cresce no máximo polinomialmente com a dimensão do problema. Por esta razão o PCV pertence à classe dos chamados problemas difíceis (*)(conceito oposto ao de problemas fáceis para os quais se conhece um bom algoritmo de resolução, isto é, um algoritmo cujo tempo de execução cresce no máximo polinomialmente com a dimensão do problema).

(*) A demonstração de que o PCV é um problema difícil pode encontrar-se em Garey and Johnson (1979).

Para abordar problemas deste tipo restam duas alternativas: ou a utilização de métodos exactos, sabendo à partida que o tempo de computação cresce exponencialmente com a dimensão do problema (no caso do PCV o número de cidades) ou a utilização de métodos heuristicos com sacrifício de garantia de obtenção de uma solução óptima.

Para o PCV existe uma imensa quantidade de métodos de qualquer um dos dois tipos. A maior parte dos métodos exactos são técnicas enumerativas que assentam em diferentes relaxações do PCV entre as quais se destacam: a afectação linear, a árvore-1 (ou arborescência-1, no caso do PCV orientado) e a programação linear.

Os algoritmos que funcionam com base na afectação linear resultam bastante bem no caso orientado. No PCV simétrico esta abordagem conduz a demasiados subcircuitos e, por consequência a soluções muito afastadas da admissibidade. Um algoritmo que funciona com base neste tipo de relaxação é o famoso método de partições e avaliações sucessivas de Little, Murty, Sweeney e Karel (1963) artigo onde aparece pela primeira vez na literatura o termo inglês com que actualmente se designam estas técnicas de enumeração implícita (branch and bound). Alguns algoritmos desenvolvidos para o problema em estudo neste trabalho resultam de adaptações deste método. Este assunto será retomado na secção seguinte. Um outro algoritmo deste tipo, considerado bastante razoável, é o de Balas e Christofides (1981) que acrescentam a esta abordagem as técnicas da Relaxação Lagrangeana.

A relaxação do PCV na árvore-1 foi desenvolvida por Held e Karp (1970,1971) e iniciou o uso da Relaxação Lagrangeana, duma forma

sistematizada na Optimimização Combinatória. Esta abordagem conduz a resultados considerados bastante razoáveis para o PCV simétrico e será aplicada ao Problema do Caixeiro Viajante Classificado neste trabalho. A relaxação da arborescência-1, correspondente relaxação para o caso orientado, é tida como inferior à da afectação.

Existem na literatura uma série de métodos baseados na Programação Linear. A utilização da PL é dificultada pela grande quantidade de restrições lineares que as formalizações de PLI do PCV contêm. Deste modo as abordagens deste tipo assentam em relaxações que não ignoram apenas as restrições de integralidade. Estes métodos tornaram-se bastante atractivos depois do trabalho de Grostchel, sumariado em Grotschel et al. (1985), que desenvolveu algumas das desigualdades lineares que caracterizam o politopo do PCV. Estas desigualdades utilizadas como planos de corte tem permitido resolver problemas de dimensões razoáveis: Grotschel (1980) resolveu um problema com 120 cidades e Crowder e Padberg (1980) uma série de PCVs Euclideanos com dimensão não superior a 100 cidades e um problema com 318 cidades (até 1986 o maior problema resolvido). Os métodos baseados na PL permitem, em geral, obter bons resultados para o PCV simétrico. A qualidade destes métodos depende claramente da aplicabilidade dos resultados de Grotschel e como estes só são válidos no caso simétrico, este tipo de abordagem não é atractivo, quando comparado com outros métodos para a resolução do PCV orientado.

Uma série de outras relaxações foram utilizadas em algoritmos para este problema como por exemplo: o problema da determinação de caminhos de dimensão n e o problema do emparelhamento.

Held e Karp (1962) desenvolveram um algoritmo de programação dinâmica capaz de resolver problemas de pequena dimensão (na prática até 13 cidades).

O maior problema resolvido, na literatura conhecida, contem 532 cidades e deve-se a Rinaldi e Padberg (1986). Este artigo contém também uma descrição do método utilizado para o efeito.

Um bom resumo sobre métodos de partições e avaliações sucessivas para a resolução do PCV, onde se encontram incluidos grande parte dos métodos exactos para a resolução deste problema pode encontrar-em Balas e Toth (1985).

Existem também imensas heurísticas para este problema. Podem classificar-se em dois grandes grupos: as construtivas e as melhorativas. As técnicas construtivas começam com uma solução não admissível que de acordo com alguma regra vai sendo alterada até constituir um circuito Hamiltoniano. Estas técnicas na sua grande maioria começam por uma cidade arbitrária, ou circuito num subconjunto de cidades, e vão agregando as restantes de acordo com algum critério. As técnicas melhorativas tem como ponto de partida uma solução admissível, a partir da qual procuram obter soluções melhores. No primeiro conjunto englobam-se dois métodos que serão utilizados neste trabalho: o Método do Vizinho mais Próximo, MVP, (nearest neighbour) e o Método da Inserção mais Afastada, MIA, (farthest insertion). Dentro das técnicas melhorativas merece espcial relevo o método r-óptimo de Lin (1965).

Uma descrição bastante completa dos métodos heurísticos para a determinação de soluções aproximadas para o PCV pode encontrar-se por exemplo em Golden et al. (1980).

Na literatura existem uma série de referências sobre aplicações interessantes deste problema (um bom resumo pode encontrar-se por exemplo em Garfinkel (1985)). No entanto, as razões da grande importância assumida por este problema, não devem procurar-se exclusivamente na riqueza destas (não existem muitos caixeiros viajantes apelando ansiosamente para a resolução do seu problema, nem outro tipo de aplicações que justifiquem por si só tal destaque). Um outro factor explicativo essencial para este lugar de relevo reside no facto de o PCV ser um problema típico de optimização combinatória funcionando como que uma "cobaia" onde se ensaiam as inovações. Tanto assim que grande parte das novidades na optimização combinatória aparecem via PCV sendo disso um bom exemplo a relaxação Lagrangeana. Elucidativo sobre este aspecto é próprio título do livro editado por Lawer et al. (1985): "The Travelling Salesman Problem - A Guided Tour of Combinatorial Optimazation".

2.2 O PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO

O Problema do Caixeiro Viajante Classificado resulta da imposição de restrições adicionais ao Problema do Caixeiro Viajante. O conjunto N, que representa as cidades está subdividido em m classes (clusters) e exige-se, para além das restrições habituais, que o percurso a determinar contemple uma visita continua às cidades de cada uma das m classes. Dito de outro modo, o caixeiro deve visitar

não só uma e uma só vez cada cidade, mas também cada subconjunto (classe) de cidades deve ser percorrido continuamente, isto é , a partir do momento em que o caixeiro se encontra numa cidade de uma determinada classe só pode passar a uma cidade de uma outra classe após ter concluído a volta às cidades dessa classe. Não existe qualquer ordenação das classes préviamente definida, deste modo este problema pode ser encarado como dois PCV simultâneos de nível hierárquico distinto: um ao nível das classes e outro ao nível das cidades. Esta ideia será desenvolvida no capítulo seguinte quando utilizada na concepção de heurísticas.

Neste problema o conjunto das arestas pode considerar-se subdividido em dois grandes conjuntos: o conjunto das arestas não locais que representam ligações entre cidades de grupos diferentes e o conjunto das arestas locais que representam ligações entre cidades de uma mesma classe. O número de arestas não locais em qualquer solução admissível será obrigatóriamente igual ao número de classes, consequência das restrições de classe.

Sendo,

$N=\{1,2,\ldots,n\}$ o conjunto de cidades a serem visitadas,

$M=\{1,2,\ldots,m\}$ o conjunto das classes,

n o número de cidades,

m o número de classes,

C uma matriz de reais cujo elemento genérico representa a distância da ligação (i,j),

G_K o conjunto de cidades que formam a classe k,

$x_{ij} = 1$ se (i,j) está na solução

0 caso contrário ,

e

|*| a cardinalidade do conjunto * ,

o Problema do Caixeiro Viajante Clássificado, pode ser formalizado, como um problema de programação linear inteira, por exemplo do seguinte modo, Lokin (1978):

$$\text{Min} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \; x_{ij} \qquad (2.1)$$

s.a

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{para } i=1,2,\ldots,n \qquad (2.2)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{para } j=1,2,\ldots,n \qquad (2.3)$$

$$\sum_{i \in N^*} \sum_{j \in N^*} x_{ij} \leqslant |N^*|-1 \qquad \text{para } \forall\, N^* \subset N,\ |N^*| \geqslant 1 \qquad (2.4)$$

$$\sum_{i \in G_K} \sum_{j \in G_K} x_{ij} = |G|-1 \qquad \text{para } G_k \subset N,\ |G_k| \geqslant 1,\ k=1,2,\ldots,m \qquad (2.5)$$

$$x_{ij} = 0 \text{ ou } 1 \qquad \text{para } i=1,2,\ldots n \text{ e } j=1,2,\ldots n \qquad (2.6)$$

A função objectivo, (2.1), traduz o facto de se pretender minimizar a distância total percorrida. Os conjuntos de restrições (2.2) e (2.3) asseguram que cada cidade seja visitada uma e uma só vez; com (2.2), impõe-se que de cada cidade só se saia uma vez e com (3) obriga-se a que a cada cidade só se chegue uma vez. As restrições (2.4) impedem a formação de subcircuitos, obrigando a que o número de arestas de ligação entre qualquer subconjunto estrito de nodos

não vazio, seja inferior à cardinalidade desse subconjunto subtraída de uma unidade não sendo possível com essa quantidade de arestas formar um circuito nesse subconjunto. As expressões (2.5) traduzem a restrição de classe, pois obrigam a que o número de arestas seja tal, que conjugado com (2.2) e (2.3) conduz à construção de um caminho Hamiltoniano nos nodos de cada classe.

A formalização que acabou de se apresentar é válida tanto para o PCVC simétrico como para o PCVC assimétrico. No capítulo 4 deste trabalho serão apresentadas outras formalizações, apenas válidas para a versão simétrica, que permitem obter duas relaxações Lagrangeanas deste problema.

O conjunto das soluções admissíveis deste problema é mais restrito do que o do PCV (a introdução das restrições de classe acarreta um a diminuição do número de soluções admissíveis de $(1/2)*(n-1)!$ para $(1/2)*(m-1)! \prod_{k=1}^{m} [(1/2)*|G_K|!]$), sendo portanto possível, em princípio, encontrar a sua solução óptima mais rápidamente. Apesar do que acabou de ser dito o PCVC parece arrastar a intractabilidade inerente ao problema do caixeiro viajante.

As referências na literatura a este problema são bastante escassas, no entanto, encontraram-se alguns métodos para resolver o Problema do Caixeiro Viajante Classificado. Chisman (1975) apresenta como sendo um procedimento razoável para problemas de pequena dimensão (com menos de 10 cidades) a resolução directamente através do algoritmo de Dantzig et al. (1954) ou de Wagner (1969), acrescentando à

formalização as restrições de classe; para problemas maiores sugere que num primeiro passo se altere a matriz das distâncias,subtraíndo uma constante, K, com um valor elevado ($K \geqslant \max c_{ij}$ para $\forall$ i,j) a todas arestas locais, de forma a que num passo seguinte a aplicação de um método de partições e avaliações sucessivas para o PCV a esta matriz transformada conduza a um circuito admissível para o PCVC. Como possiblidades para este segundo passo apresenta os algoritmos de Held e Karp (1971) e Little et al. (1963). Chisman apresenta 3 razões para que o procedimento em dois passos atrás descrito conduza a uma solução óptima admissível para o PCVC:

1- subtraíndo um número elevado, K, a todas as arestas locais estas tornam-se mais atractivas relativamente às arestas não locais;
2- como a todas as arestas locais foi subtraído um mesmo valor, K, a relação entre estas arestas mantem-se inalterada;
3- como a ligação entre classes tem que ser feita e a relação entre arestas não locais se mantem será forçada uma ligação óptima entre as classes.

Apesar das qualidades da transformação proposta, como não existe efectivamente nenhuma imposição das restrições de grupo a aplicação dos algoritmos para o PCV pode conduzir a situações de muito pouca eficácia para a obtenção de uma solução admissível para o PCVC.

Lokin (1978) utilizando um artifício equivalente ao de Chisman para transformar a matriz das distâncias (somando K às arestas não locais) chega precisamente à conclusão de que o algoritmo de Little et al. (1963) é muito pouco eficaz para resolver o problema do caixeiro viajante com a matriz alterada e chegar a uma solução admis-

sível para o PCVC. Depois de procurar a razão de tal insucesso propõe um método de partições e avaliações sucessivas, que resulta de uma adaptação do algoritmo de Little et al. (1963), onde são eliminadas as características que julga conduzirem ao insucesso acima referido, para resolver o problema do caixeiro viajante classificado orientado. Uma descrição pormenorizada deste método pode ser encontrada em Lokin (1978). O mesmo autor sugere para tratamento deste problema na versão simétrica a aplicação à matriz transformada do algoritmo de Held e Karp (1971).

Jongens e Volgenant (1985) desenvolvem um algoritmo de partições e avaliações sucessivas, apenas para o caso simétrico, resultante da adaptação de um método por eles concebido para o PCV baseado na relaxação no problema da determinação da árvore-1 mínima. Apresentam também uma heurística que utilizam para obter majorantes do valor óptimo.

2.3 APLICAÇÕES E PROBLEMAS RELACIONADOS COM O PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO.

Nesta secção reunem-se alguns exemplos de aplicações e problemas relacionados com o PCVC.

Repare-se em primeiro lugar que, em termos práticos, qualquer aplicação do problema do caixeiro viajante clássico transforma-se num PCVC se existir alguma regra de formação de grupos que contenha informação não reconvertível à matriz de distâncias. Por exemplo, o problema da determinação do percurso de um caixeiro para visitar n cidades, distribuidas por m regiões distintas, é um PCVC se tiver

como objectivo simultâneo a minimização do número de vezes que transpõe fronteiras entre regiões, dados os custos incorridos e o tempo gasto, e a minimização da distância total percorrida. Na literatura encontram-se dois exemplos interessantes de aplicações directas do PCVC que a seguir se apresentam.

Recolha de produtos num armazém, Chisman (1975).

Num armazém dispõe-se de um veículo motorizado destinado à recolha de produtos para a satisfação de encomendas. Estas encomendas contêm várias subencomendas que por sua vez se referem a diferentes tipos de produto (número de stock). Pretende-se determinar o percurso de comprimento mínimo que o veículo deve efectuar de modo a que a recolha dos produtos de uma subencomenda só se inicie após conclusão da satisfação da subencomenda anterior. Neste caso a localização de cada tipo de produto corresponde à localização de uma cidade e os tipos de produto contidos numa subencomenda formam um grupo de cidades (classe).

Foi ao estudar um problema deste tipo para a J.P. Stevens & Co Inc que Chisman encontrou esta nova classe de problemas que resolveu designar por Clustered Travelling Salesman Problem.

Ordenação de tarefas, Lokin (1978).

Uma outra aplicação interessante é a ordenação de um conjunto de n tarefas a serem executadas por uma máquina que funciona com m braços alternativos. Cada tarefa requere um determinado braço colocado na máquina. Quando se pretende que o conjunto de tarefas seja executado em tempo mínimo e que cada braço seja colocado na máquina

uma e uma só vez, o problema resultante é o do caixeiro viajante classificado em que as tarefas são as cidades e as classes são formadas pelos subconjuntos de tarefas que exigem a colocação de um mesmo braço.

Existem algumas situações concretas que conduzem a formalizações próximas do problema em estudo neste trabalho. Uma destas situações é uma pequena variante do exemplo da ordenação de tarefas apresentada também por Lokin (1978). Neste caso por razões tecnológicas impõem-se o respeito de determinadas relações de precedência entre operações (classes). O problema da ordenação deste conjunto de tarefas de forma a serem no conjunto executadas em tempo mínimo é assim um PCVC com restrições adicionais.

Uma outra situção do mesmo tipo acontece com frequência nas indústrias texteis quando se trata de recortar peças estendidas num rectângulo de tecido. Com o desenvolmento da tecnologia esta operação é realizada automáticamente, dependendo a eficiência desta etapa do processo sobretudo do percurso descrito pela ferramenta para recortar o conjunto das peças préviamente estabelecido. Este problema pode ser formulado do seguinte modo:

- Dado um conjunto de formas planas representadas por polígonos regulares ou irregulares com qualquer número de vertices assentes no interior de uma superfície rectângular, encontrar um percurso linear de comprimento mínimo que, partindo de um dos cantos do rectângulo externo, visite todas as formas sem repetição, terminando e reiniciando cada etapa do percurso num único vértice de cada uma delas, Távora (1987).

Apesar de com facilidade se reconhecerem semelhanças com o PCVC, este problema difere essencialmente em dois aspectos: na exigência de que a cidade de entrada e saída de cada classe (forma) coincidam e no facto de que uma vez estabelecido o ponto de entrada na classe o percurso dentro desta fica completamente determinado.

Podem encontrar-se na literatura uma série de referências a problemas que tal com o problema em estudo neste trabalho são variantes do PCV. Uma boa referência sobre este assunto é o trabalho de Garfinkel (1985).

Para finalizar referem-se dois problemas a exemplificar diferentes tipos de relações estreitas com o PCVC. Um exemplo interessante é o problema, habitualmente designado por Problema do Caixeiro Viajante Generalizado, onde o conjunto de cidades também está subdividido em classes exigindo-se apenas que um número mínimo de cidades em cada classe seja visitado. Um outro exemplo deve-se a Lokin (1978) e consiste na imposição de relações de precedência entre cidades, o problema em causa pode também ser considerado, tal como o PCVC, um PCV com restrições adicionais.

No capítulo seguinte discutem-se alguns métodos heurísticos que se destinam à obtenção de soluções aproximadas para o PCVC. Este tipo de soluções permitem muitas vezes em problemas intratáveis resolver situações concretas dadas as dificuldades de determinar a solução óptima.

3. MÉTODOS HEURÍSTICOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO

O propósito deste capítulo é apresentar métodos heurísticos destinados à obtenção de soluções aproximadas para o problema do caixeiro viajante classificado simétrico.

Como se referiu no capítulo anterior a literatura sobre este problema é bastante escassa tendo-se encontrado apenas um método heurístico para o PCVC em Jongens e Volgenant (1985). Este método consiste basicamente na determinação de uma solução aproximada para o PCV sujeita no final, se necessário, a algumas alterações para a tornar admissível para o PCVC. Parecendo mais indicada a aplicação de métodos logo de início destinados à obtenção de soluções para o PCVC foram desenvolvidos alguns algoritmos e procedeu-se a um estudo comparativo.

Um dos métodos concebidos resultou de uma adaptação ao problema classificado de uma heurística para o PCV (método do vizinho mais próximo). Foram ainda desenvolvidos dois métodos procurando explorar uma característica particular do PCVC. A estrutura deste problema permite encará-lo como se tratasse do enunciado de dois PCVs simultâneos de níveis hierárquicos distintos (veja-se figura 3.1).

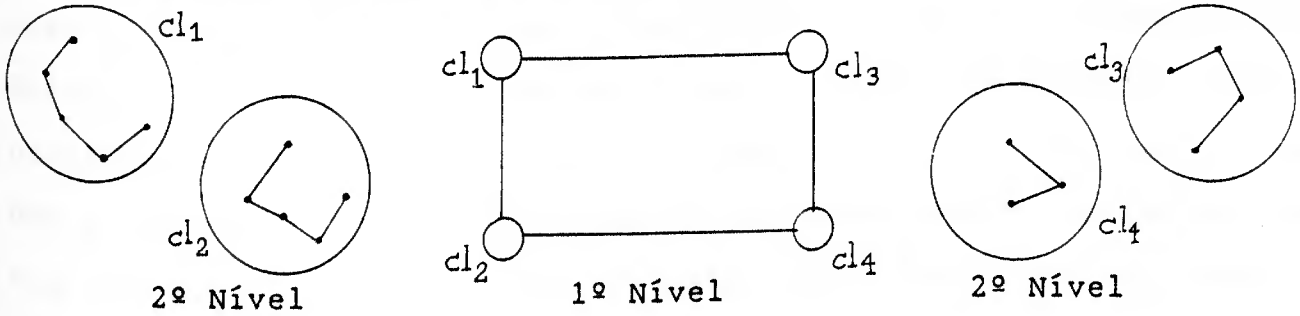


Figura 3.1

Um primeiro consiste na determinação de um circuito Hamiltoniano entre as classes,consequência das restrições de classe, que obrigam a que o número de arestas não locais seja exactamente igual ao número de classes, e das restantes restrições que obrigam à passagem por todas as classes. Um segundo nível consiste na determinação de um caminho Hamiltoniano entre as cidades de cada uma das classes. Evidentemente que a ligação entre estes dois níveis não é automática sendo necessário obviar a violações de restrições eventualmente provocadas no primeiro nível.

Como a concretização de qualquer das heurísticas para o PCVC recorre, necessariamente, a métodos para a determinação de circuitos (ou caminhos) Hamiltonianos, numa primeira secção são apresentados os algoritmos utilizados para este efeito. Numa segunda secção descrevem-se os métodos heurísticos para o PCVC. E, numa última secção, apresentam-se e analisam-se os resultados dos testes efectuados com o objectivo de estabelecer algumas comparações entre estes métodos.

3.1 MÉTODOS HEURÍSTICOS PARA A DETERMINAÇÃO DE CIRCUITOS HAMILTONIANOS

Existe na literatura uma grande quantidade de métodos heurísticos para o PCV. Uma boa referência sobre este assunto é o trabalho de Golden et al. (1980) onde se descrevem uma série de métodos e se apresenta um estudo comparativo. Contudo, no geral, não existe um bom processo de escolha entre estes métodos tendo a selecção das heurísticas a utilizar neste trabalho sido ditada, essencialmente, pela particularidade das situações que requerem a sua aplicação.

Assim optou-se pelo método do vizinho mais próximo, MVP, para determinar caminhos Hamiltonianos e pelo método da inserção mais afastada para obter um circuito Hamiltoniano nas classes.

A exposição destes métodos será feita tendo em vista a determinação de um circuito Hamiltoniano num grafo completo e não orientado G=(N,A), representando N um conjunto de cidades e A o conjunto das suas ligações, a cada uma das quais se associa um valor c_{ij} a representar a distância (peso ou custo) da ligação directa de i a j.

3.1.1 MÉTODO DO VIZINHO MAIS PRÓXIMO

O método do vizinho mais próximo, MVP, começa com uma cidade arbitrária, s. Depois selecciona a menor aresta incidente em s, (s,j), formando o caminho parcial {(s,j)}. Seguidamente vai alargando este caminho juntando-lhe em cada iteração a menor das arestas que liga a última cidade incluída a uma cidade ainda não considerada. Por último, quando todas as cidades estão incluídas, forma um circuito juntando ao caminho a aresta (s,k), que une a primeira à última cidade consideradas. Esquematicamente:

```
(*inicialização*)
  s:=cidade inicial do caminho
  i:=s
  CNV:=N-{s}
  circuito:=∅
  custo:=0
  iter:=0
```

```
(*iteração*)
  WHILE (iter<n) DO
     BEGIN
          determinar c_ij :=min{ c_il : l∈ CNV}
          circuito:=circuito U {(i,j)}
          custo:=custo+c_ij
          CNV:=CNV-{j}
          i:=j
          iter:=iter+1
     END
  circuito:=circuito U {(i,s)}
  custo:=custo+c_is
```

Como a diferentes cidades de começo podem corresponder diferentes circuitos, com o objectivo de tentar obter melhores soluções, o método pode ser executado n vezes, em cada uma das quais se começa numa cidade diferente.

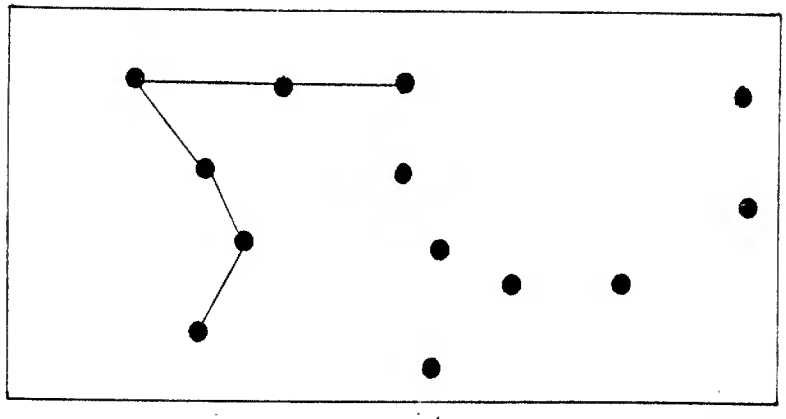
A figura 3.2 ilustra a aplicação deste método.

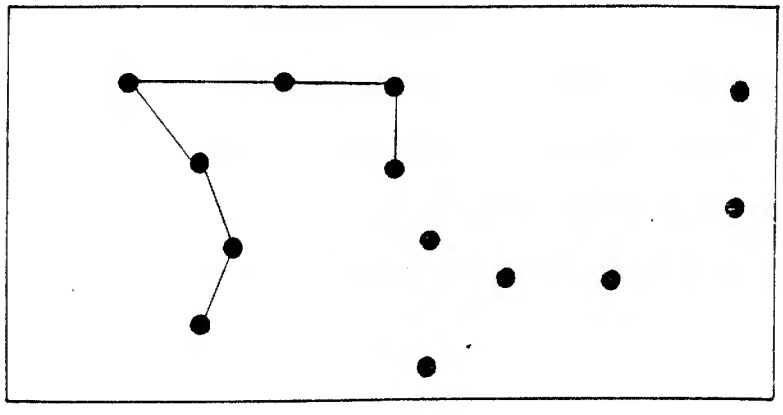
Repare-se que a determinação de caminhos Hamiltonianos com o MVP pode ser feita suprimindo o último passo, isto é, não introduzindo a última aresta (a unir a primeira à última cidade do caminho).

Este método é extremamente simples e bastante eficiente em termos computacionais, querendo com isto dizer-se que é bastante rápido e não recorre a grande espaço de memória.

Um dos defeitos que é usual apontar a este método é o modo como é

Método do Vizinho mais Próximo





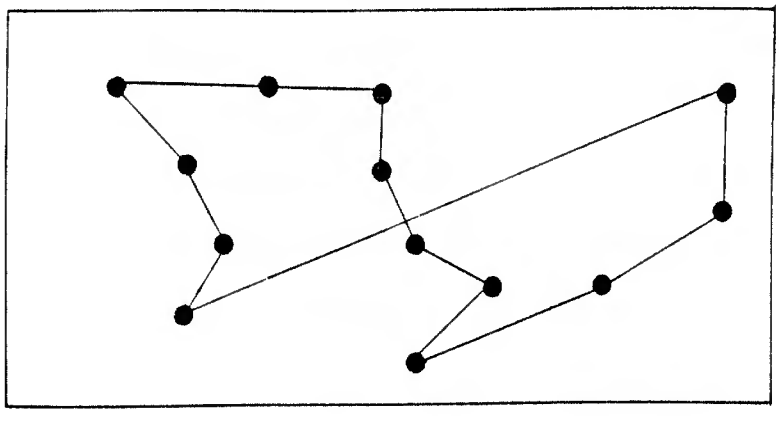


Figura 3.2

determinada a última aresta. No entanto, no caso em que a matriz das distâncias verifica a propriedade da desigualdade triangular, o peso desta aresta representa no máximo metade do valor da solução. Neste trabalho o MVP é sempre utilizado em situações nas quais a matriz de distâncias verifica esta propriedade. Para além disso, em muitos casos, é aplicado com o objectivo de determinar caminhos Hamiltonianos com a última cidade livre, não se fazendo assim sentir os efeitos do defeito acima referido.

Ainda sobre a qualidade das soluções, deve referir-se que apesar dos dos resultados de pior caso demonstrados por Rosenkrantz et al. (1974) não serem muito animadores, os testes computacionais de comparação deste método com outras heurísticas para o PCV parecem não desaconselhar convincentemente a utilização deste método, quando se avaliam conjuntamente a qualidade das soluções e a rapidez dos métodos.

3.1.2 MÉTODO DA INSERÇÃO MAIS AFASTADA

Este método começa também com uma cidade arbitrária, s. Depois selecciona a maior aresta incidente em s, (s,i), formando o circuito parcial {(s,i),(i,s)}. Seguidamente vão sendo inseridas uma a uma todas as restantes cidades no circuito. Em cada iteração a cidade a incluir é a mais afastada do circuito parcial e o local de inserção é escolhido de acordo com critérios de custo. Concretamente:

```
(*inicialização*)
 s:=cidade inicial do circuito
 i:=s
 CNV:=N-{s}
 circuito:={(s,s)}
 custo:=0
 c_ss :=0
 iter:=0
 FOR todo o u ∈ CNV  DO dist(u):=c_su

(*iteração*)
 WHILE (iter<n) DO
       BEGIN
         determinar dist(f):=Max{dist(p) : p ∈ CNV}
         FOR toda a aresta (i,j) ∈ circuito DO
                 cins_ij :=c_if +c_fj -c_ij
         (t,h):=aresta do circuito com o menor cins_th
         circuito:=circuito U {(t,f),(f,h)}-{(t,h)}
         custo:=custo+cins_th
         CNV:=CNV-{f}
         FOR todo p C CNV DO dist(p):=min{dist(p),c_fp)}
         iter:=iter+1
       END
```

O MIA pode também ser aplicado n vezes começando de cada vez numa cidade diferente.

Este método começa por inserir as cidades mais afastadas, sendo primeiro construído um contorno do conjunto de todas as cidades,

deixando para o final ,quando já há menos maleabilidade, pequenos detalhes da inserção de cidades já próximas do circuito e portanto com poucas possibilidades de penalizar muito a distância total (veja-se a figura 3.3).

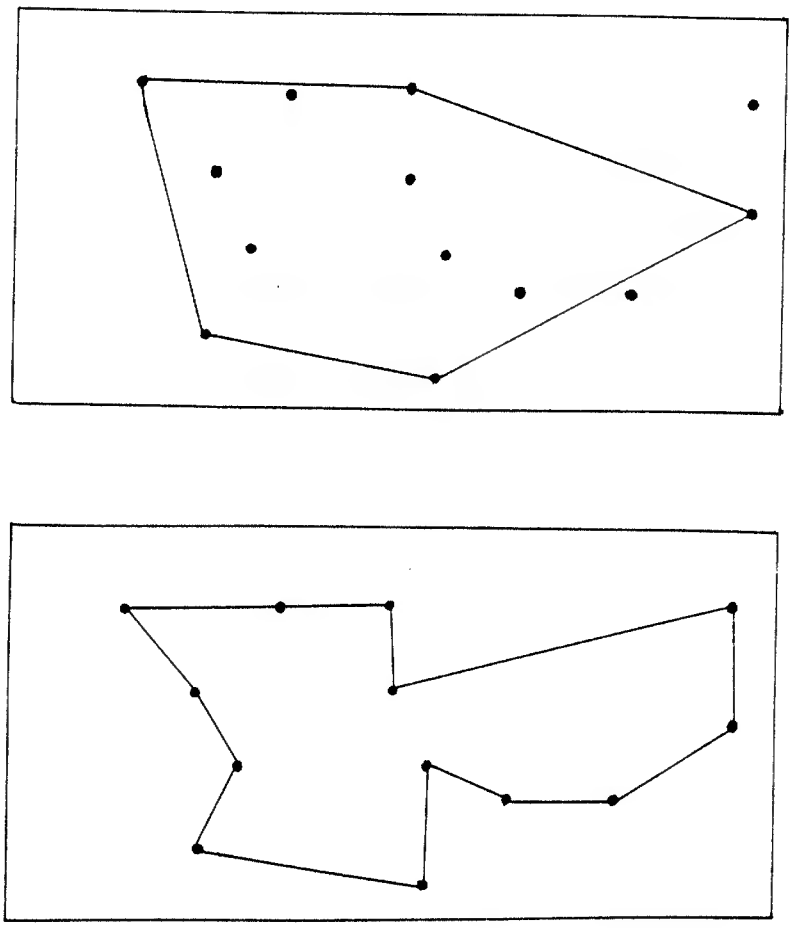


Figura 3.3

Os bons resultados obtidos com este método sugerem que a estratégia de inserir em cada passo a cidade mais afastasda do circuito, não é pior do que outras regras de selecção da cidade a inserir veja-se por exemplo Rosenkrantz et al.(1974).

3.2 MÉTODOS HEURÍSTICOS PARA O PCVC

3.2.1 MÉTODO DO VIZINHO MAIS PRÓXIMO CLASSIFICADO

Este método, na sequência referido com a sigla MVPC, consiste numa adaptação do método do vizinho mais próximo ao problema do caixeiro viajante classificado. A modificação introduzida restringe o conjunto em que a próxima cidade a ser incluída no caminho parcial é escolhida, ao conjunto das cidades pertencentes à classe da última cidade inserida no circuito e ainda não incluidas, quando esse conjunto é não vazio. Este método pode ser descrito do seguinte modo:

(CL é um vector de dimensão n que tem na posição i a classe a que pertence a cidade i)

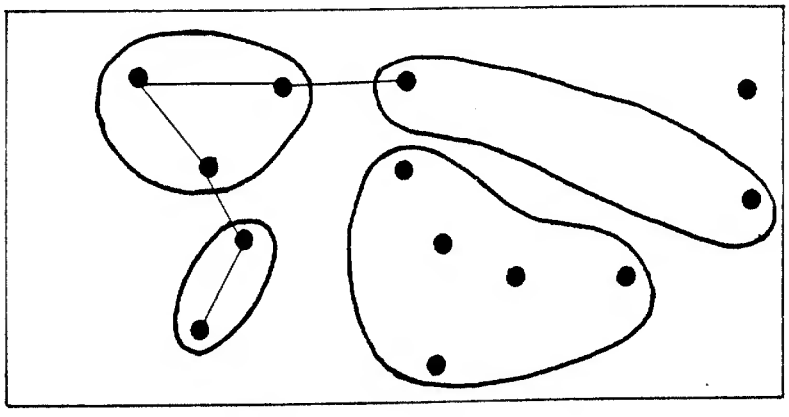
```
(*inicialização*)
  s:=cidade inicial do circuito
  i:=s
  CLNV:={1,...,m} - {CL[i]}
  iter:=0
  custo:=0
  circuito:=∅
(*iteração*)
  WHILE (iter≤m) DO
       BEGIN
          determinar um caminho Hamiltoniano em CL[i] com
                                                 inicio em i
          j:=última cidade do caminho
          circuito:=circuito∪{arestas do caminho}
          custo:=custo+custo do caminho
          iter:=iter+1
```

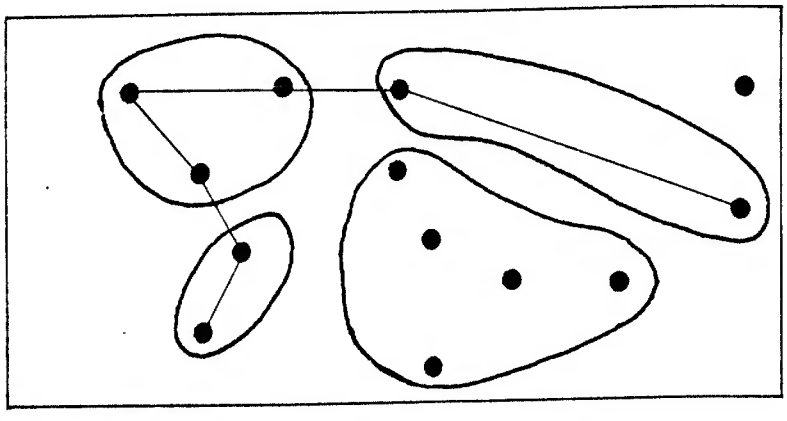
```
IF (iter<m) THEN
   BEGIN
     determinar a aresta (j,q) tal que:
                  c_jq = min{ c_jp : CL[p] ∈ CLNV}
     circuito:=circuito U {(j,q)}
     custo:=custo+c_jq
     i:=q
     CLNV:=CLNV-{CL[i]}
   END
           ELSE
   BEGIN
     circuito:=circuito U {(j,s)}
     custo:=custo+c_js
   END
END
```

Este método foi aplicado n vezes, começando de cada vez numa cidade diferente, seleccionando-se a melhor solução.

As figuras 3.4 a) e 3.4 b) ilustram a aplicação deste método e do MVP permitindo detectar as diferenças existentes. Note-se que o MVP pode não conduzir a uma solução admissível para o PCVC como aliás acontece neste exemplo.

Método do Vizinho mais Próximo Classificado





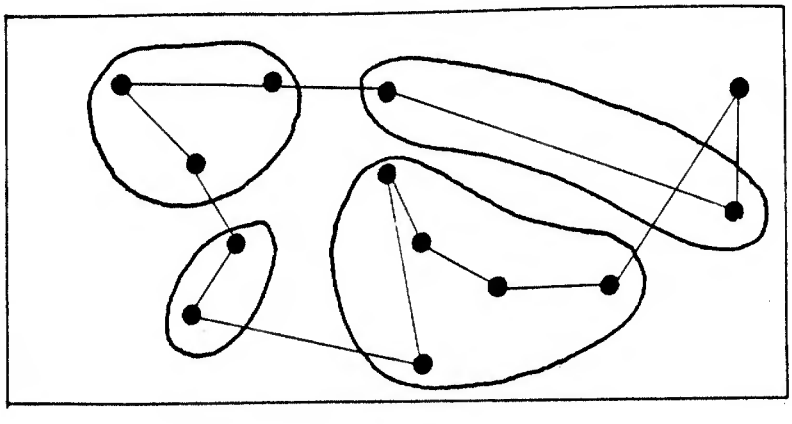
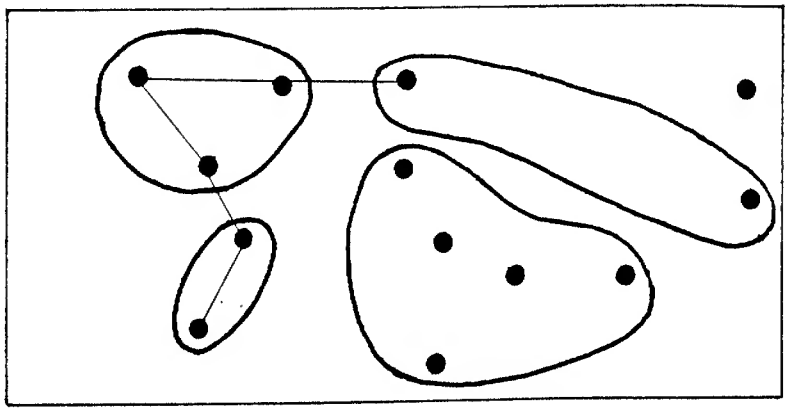
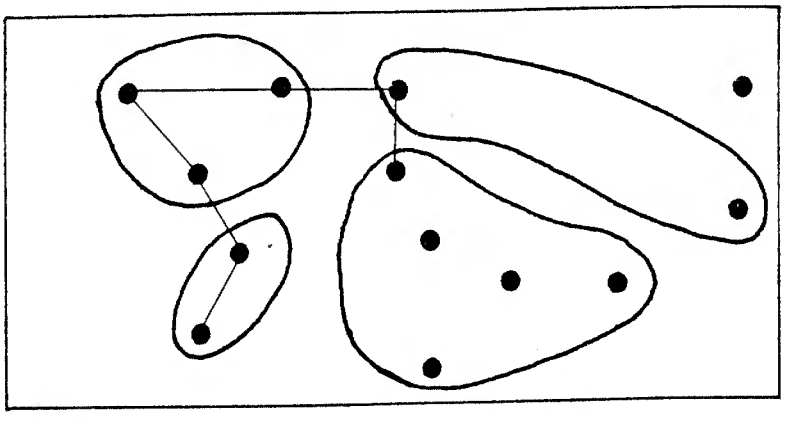


Figura 3.4 a)

Método do Vizinho mais Próximo





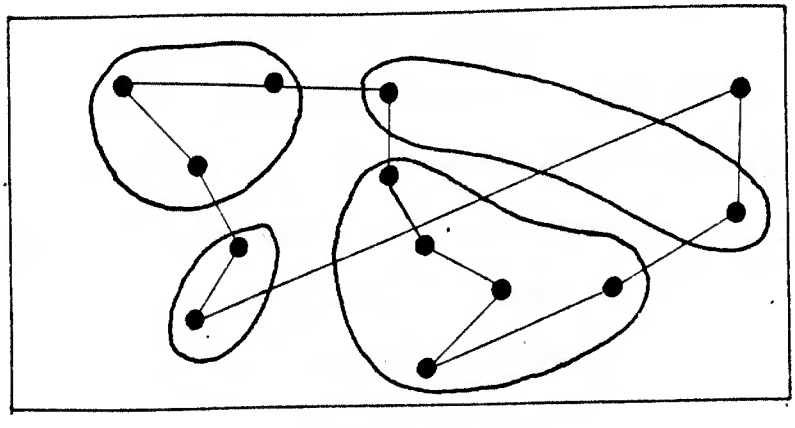


Figura 3.4 b)

3.2.2 MÉTODO CLASSES-CIDADES

Com este método, na sequência designado pela sigla MCLCI, pretende-se explorar a característica particular do PCVC referida no início deste capítulo.

Considera-se um novo grafo, designado por grafo das classes, onde cada nodo representa uma classe. A aresta (l,k) deste novo grafo, que representa a ligação entre a classe l e a classe k, é a menor das arestas que une uma cidade da classe l a uma cidade da classe k. A distância associada a cada uma das arestas é a distância da aresta do grafo inicial que ela representa. Ou seja, a matriz das distâncias do grafo das classes obtem-se a partir da matriz das distâncias do grafo inicial fazendo

$$d_{lK} = \begin{cases} \min \{c_{ij} : i \in \text{classe } l,\ j \in \text{classe } k\} & l \neq k \\ +\infty & l \neq k \end{cases}$$

O primeiro passo deste método consiste precisamente na determinação de um circuito Hamiltoniano no grafo das classes, obtendo-se deste modo um conjunto de m arestas não locais. O conjunto de arestas assim determinado não é necessariamente compatível pois pode acontecer que as duas arestas incidentes numa certa classe incidam precisamente na mesma cidade, situação que só é admissível se a classe em questão for singular. Numa segunda fase resolvem-se estas situações de incompatibilidade analisando cada uma das m classes e,quando numa certa classe k, não singular, ambas as arestas não locais, (i,p) e (i,q), incidem numa cidade i, investigam-se as possibilidades de alteração (dentro da classe em estudo) seleccionando-se a de

menor custo, isto é, determina-se

$c_{rp}=\min\{c_{jp}$ para $j \neq i \in$ classe $k\}$ e

$c_{sq}=\min\{c_{jq}$ para $j \neq i \in$ classe $k\}$

e substitui-se (i,p) por (r,p) se $c_{rp}-c_{ip} < c_{sq}-c_{iq}$ ou (i,q) por (s,q) no caso contrário.

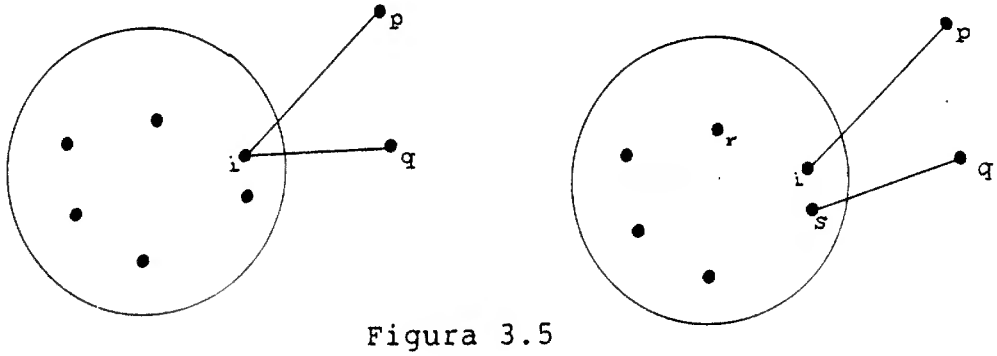


Figura 3.5

Por último, constroi-se um caminho Hamiltoniano em cada uma das classes não singulares a ligar as duas cidades de comunicação com o exterior fixadas anteriormente. Concretamente este método consiste na seguinte sequência de passos:

(CL é,como anteriormente, um vector de dimensão n que na posição i contém a classe a que pertence a cidade i; grau é um vector de dimensão n que em cada iteração contem na posição i o número de arestas do circuito corrente que incidem na cidade i)

(* determinar m arestas não locais *)

(* construir a matriz das distâncias do grafo das classes, D *)

FOR k:=1 TO (m-1) DO

FOR l:=k+1 TO m DO

d_{lk} :=min{c_{ij} para ∀ (i,j) com CL[i]=k e CL[j]=l}

Determinar um circuito Hamiltoniano no grafo das classes

circuito:={arestas não locais correspontes}

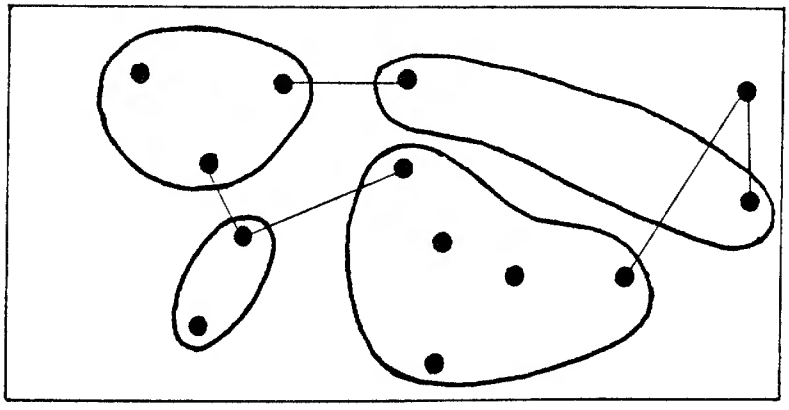
custo:=custo do circuito

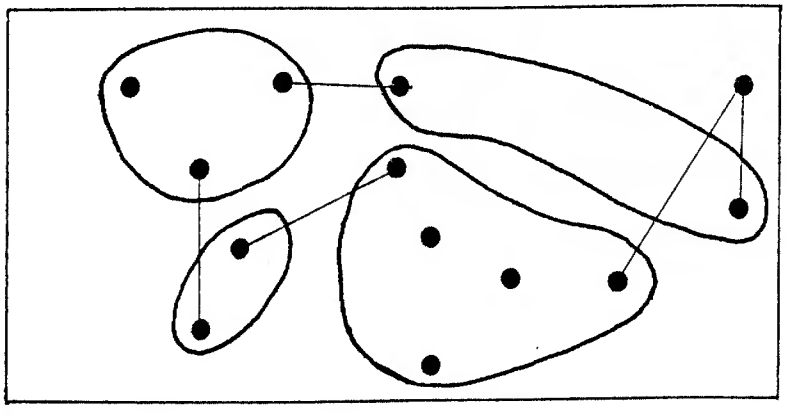
```
(*resolver eventuais situações de dupla utilização de uma cidade
  pertencente a uma classe não singular*)
  FOR i:=1 TO n DO
  IF (grau[i]=2) and (|CL[i]|>1) THEN
            BEGIN
              (i,p), (i,q) as duas arestas incidentes em i e
                                                    CL[i]=k
              c_rp:=min{c_jp para CL[j]=k e j ≠ i}
              c_sq:=min{c_jq para CL[j]=k e j ≠ i}
              IF (c_rp-c_ip)<(c_sq-c_iq) THEN
                    BEGIN
                      circuito:=circuito U {(r,p)} - {(i,p)}
                      custo:=custo+c_rp-c_ip
                    END
                                  ELSE
                    BEGIN
                      circuito:=circuito U {(s,q)} - {(i,q)}
                      custo:=custo+c_sq-c_iq
                    END

(*Determinar um caminho Hamiltoniano em cada uma das classes não
  singulares a ligar as cidades de comunicação com o exterior*)

  FOR k:=1 to M DO
      IF (|classe k|>1) THEN
                    BEGIN
                      i e j cidades da classe k com grau 1
                      determinar um caminho Hamiltoniano na
                          classe k a ligar as cidades i e j
                      circuito:=circuito U {arestas do caminho}
                      custo:=custo+{custo do caminho}
                    END
```

Determinou-se o circuito Hamiltoniano no grafo das classses utilizando o MIA, pois de certo modo tratava-se de estabelecer um contorno do percurso. Este método foi aplicado m vezes, começando de cada vez numa classe diferente seleccionando-se posteriormente a melhor das soluções. A determinação dos caminhos Hamiltonianos foi feita aplicando o MVP. Em cada classe o método foi executado começando em cada uma das cidades de comunicação com o exterior escolhendo-se depois das duas soluções obtidas a melhor.

Na figura 3.6 ilustram-se as principais fases do MCLCI

Método Classes-Cidades





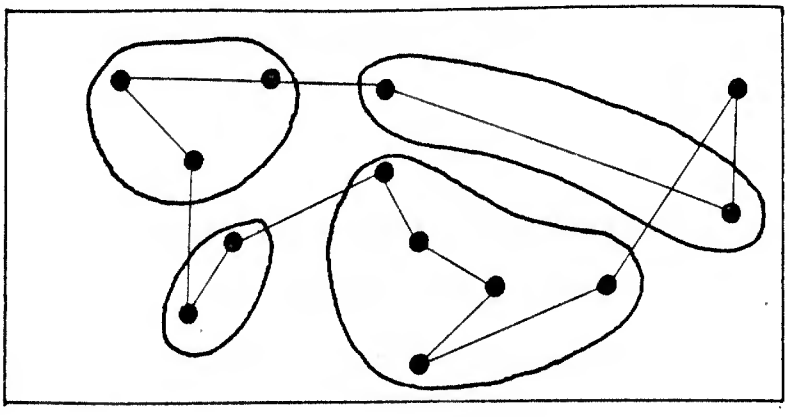


Figura 3.6

3.2.3 MÉTODO CIDADES-CLASSES

Este método, no seguimento referido com a sigla MCICL, considera a mesma decomposição de problemas do método anterior mas por ordem inversa.

Numa primeira etapa determina-se em cada classe um caminho Hamiltoniano a ligar as respectivas cidades. Obtêm-se deste modo as n-m arestas locais e fixam-se as cidades de comunicação com o exterior de cada classe. Num passo seguinte constroi-se um grafo das classes de forma análoga ao do método anterior mas agora a aresta (l,k) (que representa a ligação entre as classes l e k) é a menor das arestas do grafo inicial que une uma cidade de comunicação com o exterior da classe l com uma cidade de comunicação com o exterior da classe k, com $l \neq k$, evidentemente. A distância associada a cada uma das arestas do novo grafo, d_{lK} , é a distância da aresta do grafo inicial, isto é

$$d_{lK} = \min \{c_{ip}, c_{iq}, c_{jp}, c_{jq}\},$$

onde i e j são as cidades terminais do caminho Hamiltoniano da classe l e, p e q são as cidades terminais do caminho Hamiltoniano da classe k. Determina-se então um circuito Hamiltoniano no grafo das classes obtendo-se deste modo um conjunto de m arestas não locais.

Estão encontradas n arestas das quais exactamente m são não locais, no entanto, podem existir classes em que o grau das cidades de ligação com o exterior seja diferentes de dois. É portanto necessário ainda um último passo para a correcção destas situações. Estas

correcções são feitas classe a classe seleccionando-se a alternativa de menor custo. Concretamente admita-se que a cidade i da classe l tem grau 3 e portanto a cidade j, a outra cidade terminal da classe l, tem grau 1 admita-se ainda que as arestas não locais incidentes na classe l são (i,p) e (i,q), substituir-se-à (i,p) por (j,p) se $c_{jp}-c_{ip} < c_{jq}-c_{iq}$ ou (i,q) por (j,q) caso contrário.

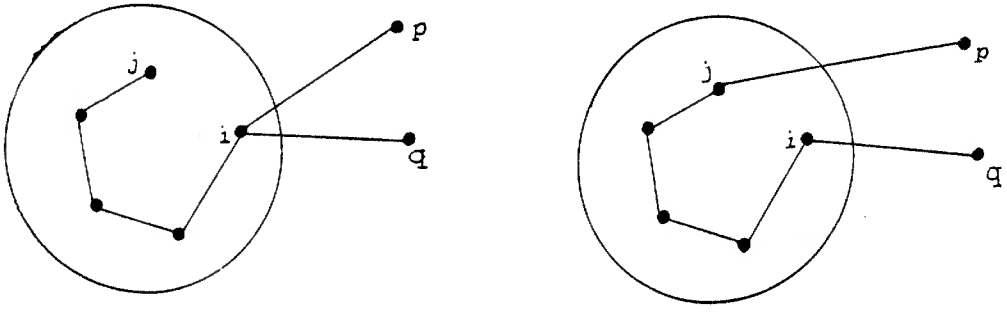


Figura 3.7

Em pseudo-código:

```
( CL é um vector definido como nos métodos anteriores )
  (*inicialização*)
    circuito:=∅
    custo:=0
  (*iteração*)
    (*determinação de caminhos Hamiltonianos nas classes*)
    FOR k:=1 TO m DO
        BEGIN
          determinar um caminho Hamiltoniano a ligar as cidades
                                                  da classe k
          circuito:=circuito U {arestas do caminho}
          custo:=custo+custo do caminho
        END
```

```
(*construir da matriz das distâncias do grafo das classes,D*)
FOR k:=1 TO m-1 DO
     FOR l:=k+1 TO m DO
          BEGIN
            i e j cidades terminais do caminho da classe k
            p e q cidades terminais do caminho da classe l
            d_lK :=min{c_ip ,c_iq ,c_jp ,c_jq }
          END
(*determinar m arestas não locais*)
determinar um circuito Hamiltoniano no grafo das classes
custo:=custo+custo do circuito das classes
circuito:=circuito U {arestas não locais  corespondentes ao  cir-
                                                  cuito das classes}

FOR i:=1 TO n DO
     IF (grau[i]=3) THEN
          BEGIN
            j:=cidade da classe CL[i] com grau 1
            (i,p) e (i,q) arestas não locais incidentes em i
            IF (c_jp -c_ip <c_jq -c_iq ) THEN
                 BEGIN
                   circuito:=circuito U {(j,p)} - {(i,p)}
                   custo:=custo+c_jp -c_ip
                 END
                              ELSE
                 BEGIN
                   circuito:=circuito U {(j,q)} - {(i,q)}
                   custo:=custo+c_jq -c_iq
                 END
```

A determinação dos caminhos Hamiltonianos foi feita através do método do vizinho mais próximo. Em cada classe este método foi aplicado começando em cada uma das cidades da classe escolhendo-se depois a melhor solução. Para determinação do circuito Hamiltoniano no grafo das classes utilizou-se o método da inserção mais afastada

O MIA foi aplicado m vezes, começando de cada vez numa classe diferente escolhendo-se a melhor destas soluções.

A figura 3.9 ilustra a aplicação deste método.

3.2.4 MÉTODO DE JONGENS E VOLGENANT

Este método, na sequência referido com a sigla MJV, apresentado por Jongens e Volgenant(1985) foi o único método heurístico para o PCVC encontrado na literatura. O MJV decompõe-se basicamente em duas fases. A primeira fase consiste na determinação de um circuito Hamiltoniano ignorando-se completamente a existência de restrições de classe. A segunda fase tem como objectivo a detecção de restrições de classe que tenham eventualmente sido violadas e a sua imposição. Esta fase consiste na determinação de uma sucessão de circuitos Hamiltonianos cada um dos quais com um número de arestas não locais inferior ao anterior até se obter um circuito com exactamente m arestas não locais. Os autores justificam a aplicabilidade do procedimento com a afirmação de que um circuito Hamiltoniano ou é solução admissível do PCVC ou tem uma ou mais classes com grau par pelo menos não inferior a 4.

Método Cidades-Classes

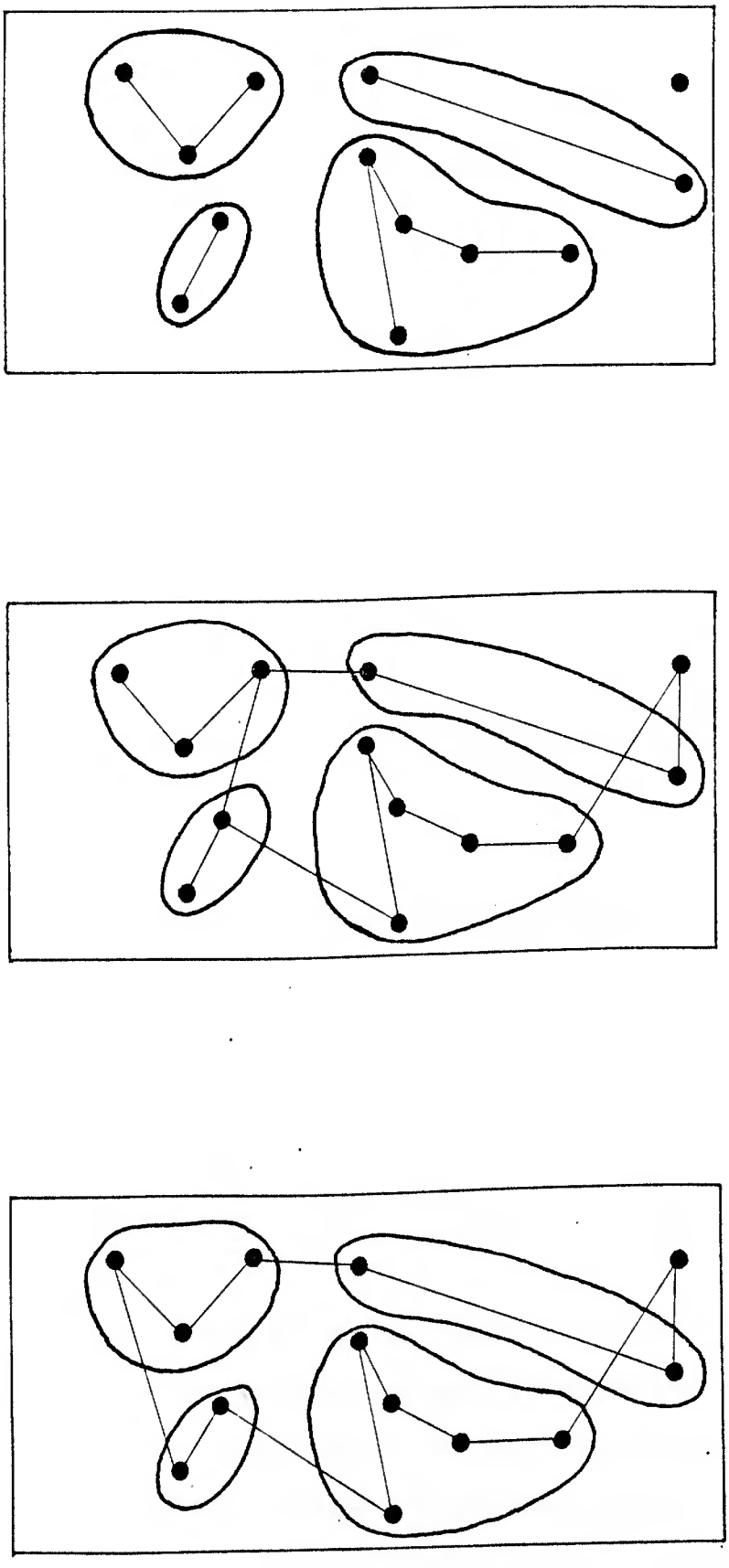


Figura 3.9

O primeiro passo de cada iteração da segunda fase consiste em seleccionar uma classe, k, com um grau não local não inferior a quatro. Num segundo passo seleccionam-se 4 arestas não locais incidentes na classe k, sejam e_1, e_2, e_3 e e_4 com $e_i=(a_i,b_i)$ e a_i pertencentes à classe k, pela ordem em que aparecem no circuito (veja-se figura 3.10).

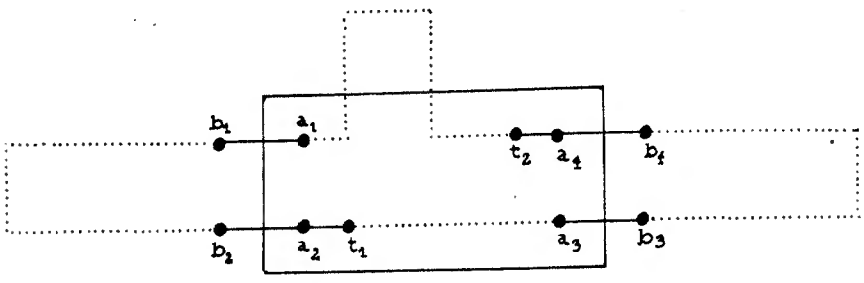


Figura 3.10

Considerem-se ainda as arestas (a_2,t_1) e (a_4,t_2) com $t_1 \neq b_2$ e $t_2 \neq b_4$.

Seguidamente observam-se as seguintes alternativas de trocas de arestas:

[A1] e_1 e e_3 por (a_1,a_3) e (b_1,b_3)

[A2] e_2 e e_4 por (a_2,a_4) e (b_2,b_4)

[A3] e_1, e_4 e (a_2,t_1) por (a_2,a_4), (a_1,t_1) e (b_1,b_4)

[A4] e_2, e_3 e (a_4,t_2) por (a_2,a_4), (a_3,t_2) e (b_2,b_3).

Selecciona-se a alternativa de menor custo com b_i e b_j pertencentes à mesma classe, caso não exista nenhuma alternativa nestas condições selecciona-se a de menor custo. A preferência de alternativas com b_i e b_j pertencentes à mesma classe deve-se ao facto de deste modo se reduzirem simultânemente o excesso de arestas não locais incidentes em duas classes.

Seguidamente apresenta-se o método em pseudo-código tal como foi utilizado neste trabalho. O significado de CL é o definido anteriormente.

(* primeira fase *)

determinar um circuito Hamiltoniano

circuito:={arestas do circuito}

custo:=custo do circuito

CLVR:={classes nas quais incidem 4 ou mais arestas não locais}

(* segunda fase *)

WHILE CLVR <> $\emptyset$ DO

BEGIN

k:=classe $\in$ CLVR com menor número de arestas não locais

e_i :=(a_i,b_i) 4 arestas não locais com CL[a_i]=k, pela ordem em que aparecem no circuito.

(a_2,t_1):=aresta incidente em a_2 com $t_1 \neq b_2$

(a_4,t_2):=aresta incidente em a_4 com $t_2 \neq b_4$

alt[1]:={(a_1,a_3),(b_1,b_3)}-{(a_1,b_1),(a_3,b_3)}

calt[1]:= $c_{a_1 a_3} + c_{b_1 b_3} - c_{a_1 b_1} - c_{a_3 b_3}$

alt[2]:={(a_2,a_4),(b_2,b_4)}-{(a_2,b_2),(a_4,b_4)}

calt[2]:= $c_{a_2 a_4} + c_{b_2 b_4} - c_{a_2 b_2} - c_{a_4 b_4}$

alt[3]:={(a_2,a_4),(a_1,t_1),(b_1,b_4)}-{(a_1,b_1),(a_4,b_4),(a_2,t_1)}

calt[3]:= $c_{a_2 a_4} + c_{a_1 t_1} + c_{b_1 b_4} - c_{a_1 b_1} - c_{a_4 b_4} - c_{a_2 t_1}$

alt[4]:={(a_2,a_4),(a_3,t_2),(b_2,b_3)}-{(a_2,b_2),(a_3,b_3),(a_4,t_2)}

calt[4]:= $c_{a_2 a_4} + c_{a_3 t_2} + c_{b_2 b_3} - c_{a_2 b_2} - c_{a_3 b_3} - c_{a_4 t_2}$

REDSIM:= {alternativas com uma aresta (b_i,b_j) na qual CL[b_i]=CL[b_j] }

IF REDSIM $\neq \emptyset$ THEN s:=min{ calt[l] :l $\in$ REDSIM}

ELSE s:=min{ calt[l] : l =1,2,3,4}

```
        circuito:=circuito U { alt [s] }
        custo:=custo + calt[s]

        CLVR:={classes nas quais incidem 4 ou mais arestas
                              não locais, no circuito corrente}
    END
```

Na figura seguinte ilustram-se as duas fases de aplicação do MJV.

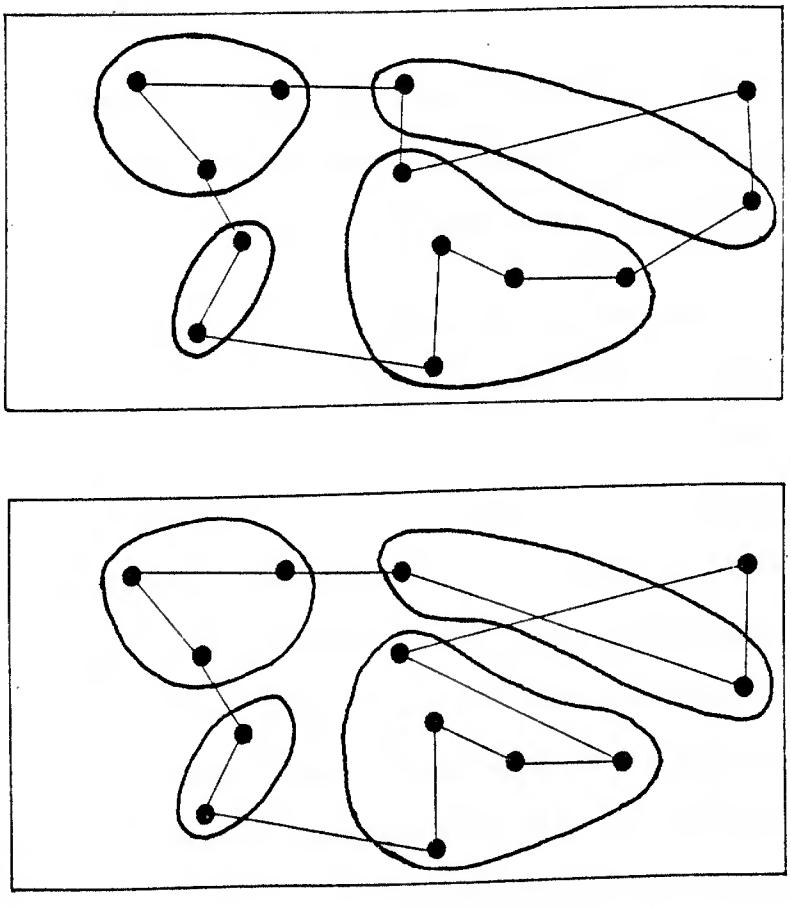


Figura 3.11

Jongens e Volgenant não explicitam qual o método que utilizaram para determinar o caminho Hamiltoniano da primeira fase, tendo-se neste trabalho optado por escolher o melhor de n aplicações do MIA

e n aplicações do MVP, pois pretende-se elaborar um estudo comparativo deste método com os anteriormente apresentados.

Na secção seguinte apresentam-se os resultados de trabalho computacional realizado com o objectivo de comparar estes métodos heurísticos.

3.3 RESULTADOS COMPUTACIONAIS

Nesta secção analisam-se os resultados de testes efectuados com o objectivo de tentar estabelecer algumas comparações sobre os métodos heurísticos para o PCVC apresentados na secção anterior.

Estes resultados referem-se à aplicação dos métodos em estudo, programados em Pascal, a um conjunto de 73 problemas. Neste conjunto inclui-se o único problema cujos dados estão publicados na literatura. Este problema com 24 cidades e 7 classes deve-se a Lokin (1978) e corresponde a um exemplo concreto de ordenação de tarefas. Dada esta escassez de dados e como se pretendia tirar conclusões sobre o desempenho dos referidos métodos na presença de problemas a representarem configurações espaciais distintas, correspondentes a diferentes aplicações, foram gerados 24 problemas para cada uma de 3 estruturas de dados com as seguintes particularidades:

DT0 - cidades de classes diferentes podem ter a mesma localização, como no problema de Lokin;

DT1 - as classes são definidas independentemente da proximidade das cidades que contêm;

DT2 - as distâncias entre cidades de uma mesma classe são majoradas por uma constante (pequena relativamente à

distância máxima admitida entre duas cidades de classes diferentes).

Procurando-se proporcionar ainda uma análise do efeito da dimensão, isto é, do número de cidades e de classes dos problemas nos diversos métodos, foram gerados 9 problemas para cada uma das seguintes 8 dimensões: 80 cidades com 6 e 12 classes, 100 cidades com 8 e 15 classes, 120 cidades com 9 e 18 classes e 150 cidades com 11 e 22 classes. Estas dimensões, utilizadas por Jongens e Volgenant (1985) para testar o seu algoritmo para a resolução do PCVC, foram definidas de modo a permitir também a detecção de efeitos do aumento do número de classes em problemas com igual número de cidades. Os detalhes sobre o modo de geração dos problemas descrevem-se em anexo.

A aplicação dos métodos em estudo foi então efectuada neste conjunto de problemas tendo em atenção as características essenciais em discussão.

As propriedades mais importantes dos métodos heurísticos referem-se à qualidade das soluções que permitem obter e à rapidez com que o fazem. Quanto ao primeiro aspecto, não se dispondo de qualquer informação sobre o valor óptimo, os métodos podem ser comparados analisando-se os resultados sob vários pontos de vista. Por um lado, relativamente ao número de vezes que cada um deles obtem a melhor solução (isto é, a solução de valor mais baixo) e,por outro, observando a média dos desvios percentuais relativamente à melhor solução obtida. Mais adiante poderá fazer-se ainda um outro estudo em função dos minorantes que se determinam no capítulo seguinte. Quanto à rapidez, os métodos podem ser comparados analisando a média dos tempos de CPU gastos na execução de cada um deles.

No quadro seguinte apresentam-se os resultados referentes ao número de vezes que cada método apresentou a melhor solução agrupando-se os problemas de igual dimensão.

nº cidades	nº classes	MCLCI	MCICL	MVPC	MJV
24	7	0	0	1	0
80	6	0	3	3	3
	12	0	1	5	3
100	8	0	3	4	2
	15	0	1	5	3
120	9	3	0	6	0
	18	3	0	6	0
150	11	1	3	5	0
	22	1	1	7	0
TOTAL		8	12	42	11

Um aspecto curioso a salientar é a fraca sensibilidade dos métodos sob este ponto de vista a diferentes proporções número de cidades/número de classes.

Da análise deste quadro, podem retirar-se as seguintes conclusões relativamente ao conjunto dos 73 problemas testados:

- em termos globais o melhor método é o MVPC;
- para problemas com dimensão superior a 100 cidades o MJV deixa de ser competitivo com os outros métodos;
- para problemas com dimensão inferior a 120 cidades o MCLCI é um método claramente inferior aos restantes.

Estas conclusões são de certo modo clarificadas se se organizarem os resultados por tipo de dados. É o que se faz no quadro seguinte:

	100 cidades ou menos			120 cidades ou mais		
tipo de dados	DT0	DT1	DT2	DT0	DT1	DT2
MCLCI	0	0	0	0	0	8
MCICL	2	5	1	1	0	3
MVPC	11	7	0	11	12	1
MJV	0	0	11	0	0	0

Da leitura deste quadro torna-se evidente, para o conjunto dos problemas testados, que a qualidade comparativa dos métodos é notoriamente afectada pela estrutura dos dados. Quando os dados do problema se referem a uma estrutura espacial efectivamente com configuração de classes, isto é, quando as cidades de cada classe estão próximas entre si, como acontece com os dados do tipo DT2, o MJV é de entre os quatro métodos o que permite obter melhores soluções nos problemas com uma dimensão não superior a 100 cidades. A partir desta dimensão o método que, regra geral, dá melhores resultados com este tipo de dados é o MCLCI.

As virtudes do MJV na situação acima referida, devem procurar-se na qualidade do circuito Hamiltoniano de partida que, pela configuração dos dados não é muito afastado de um circuito admissível para o problema do caixeiro viajante classificado.

O MVPC é, regra geral, o melhor dos métodos nos outros tipos de dados, sendo a sua superioridade mais evidente nos dados do tipo DT0.

Para completar o estudo comparativo dos métodos relativamente à qualidade das soluções, analisam-se ainda os quadros seguintes que referem as médias das percentagens dos desvios relativamente à melhor solução obtida em cada problema.

nº cidades	nº classes	CLCI	CICL	MVPC	MJV
24	7	14.922	9.691	0	13.847
80	6	10.150	1.742	1.756	40.895
	12	11.225	3.488	3.172	35.228
100	8	9.193	1.691	1.752	49.124
	15	12.582	5.999	2.533	35.086
120	9	10.021	3.277	1.433	53.872
	18	11.038	5.381	1.636	36.939
150	11	9.637	1.522	1.186	63.167
	22	12.373	4.325	0.849	35.433
	MÉDIA	10.834	3.514	1.790	43.309

	CLCI	CICL	MVPC	MJV
DT0	14.499	4.406	0.214	68.666
DT1	14.819	3.952	0.304	54.941
DT2	2.033	2.145	4.851	5.264
MÉDIA	10.834	3.514	1.790	43.309

Da observação destes valores ressaltam dois aspectos interessantes. O primeiro refere-se à sensibilidade da totalidade dos métodos a diferentes proporções número de cidades/número de classes. No entanto, enquanto a qualidade das soluções obtidas com o MJV melhora com aumento do número de classes, nos restantes métodos verifica-se precisamente o inverso. O segundo aspecto refere-se à possibilidade de ordenação dos métodos por ordem decrescente da qualidade das so-

luções obtidas do seguinte modo:

- MVPC
- MCICL
- MCLCI
- MJV

No quadro seguinte apresentam-se as médias do tempo de CPU gasto na aplicação de cada um dos métodos, programados em Pascal, num VAX 780 (em segundos). Nestes tempos incluem-se quer a leitura de ficheiros de dados quer a execução dos métodos propriamente dita. Deve ainda referir-se que os resultados são afectados pelo modo como se resolveram os pormenores de implementação, existindo eventualmente outras soluções conducentes a melhores resultados.

nº cidades	nº classes	MCLCI	MCICL	MVPC	MJV
24	7	0.67	0.58	0.71	3.770
80	6	3.814	4.260	7.543	127.530
	12	4.121	4.067	7.671	127.206
100	8	5.707	6.326	12.213	246.936
	15	6.450	6.537	12.762	246.544
120	9	8.391	9.816	18.247	424.121
	18	9.414	9.958	20.561	424.416
150	11	12.314	14.300	30.767	834.037
	22	14.471	15.249	34.399	834.625
MÉDIA		8.085	8.814	18.020	408.177

Face aos resultados, os métodos em estudo podem ser ordenados por ordem decrescente de rapidez do seguinte modo:

- MCLCI
- MCICL
- MVPC
- MJV.

Deve salientar-se a existência de grandes diferenças entre os tempos destes dois primeiros métodos (muito semelhantes e significativamente mais baixos) e o último método, MJV, de longe o mais pesado. No entanto, o tempo gasto na segunda fase do MJV é bastante reduzido comparado com o tempo gasto na obtenção do circuito Hamiltoniano de partida que envolve a execução de n MVPs e n MIAs. Aliás os tempos de computação dos diversos métodos seriam certamente reduzidos se não se tivesse optado sistemáticamente, como já se referiu, por efectuar um grande número de corridas do MVP e do MIA para se escolher a melhor solução. A esta redução de tempo estariam naturalmente associadas piores soluções, um trabalho interessante a desenvolver consiste precisamente no estudo da relação entre estes dois efeitos.

Para finalizar, deve acrescentar-se que a experimentação de diferentes métodos para a determinação de circuitos e caminhos Hamiltonianos na concretização dos diferentes passos dos diversos métodos heurísticos para o PCVC pode aconselhar a utilização de outros métodos (alternativamente ao MVP e ao MIA) e, deste modo, a hierarquização dos métodos em estudo sugerida pelos testes efectuados ser alterada.

4. DUAS RELAXAÇÕES PARA O PROBLEMA DO CAIXEIRO VIAJANTE CLASSIFICADO

4.1 ALGUNS CONCEITOS E RESULTADOS DA DUALIDADE LAGRANGEANA

O objectivo desta secção é apresentar os conceitos e resultados da dualidade Lagrangeana que serão utilizados na secção seguinte para apresentar duas formas para a determinação de minorantes para o valor óptimo do PCVC simétrico. As provas dos resultados que se apresentam nesta secção podem encontrar-se, na sua grande maioria em Shapiro (1979).

No trabalho de Geoffrion (1974) encontra-se uma primeira sistematização das técnicas da relaxação Lagrangeana em optimização discreta. Neste artigo define-se relaxação de um problema seguinte do modo:

DEFINIÇÃO

Um problema de minimização, (Q), é uma relaxação de um problema de minimização, (P),se o conjunto das soluções admissíveis de Q, F(Q), contem o conjunto das soluções admissiveis de P, F(P), e a função objectivo de (Q) é inferior ou igual à função objectivo de (P) em F(P).

A relaxação é uma tentativa de simplificação de um problema através do afrouxamento de um determinado subconjunto das suas restrições que se entende responsável pela dificuldade de resolução do problema. Desta forma constrói-se um novo problema que permite obter minorantes (ou majorantes, se se tratar de um problema de maximização) para o valor do óptimo do problema em estudo.

Considere-se o seguinte problema:

$$z = \min f(x)$$
$$\text{s.a} \quad g(x) \leqslant 0 \qquad (P)$$
$$x \in X$$

com X compacto e, f e g funções contínuas de R^n em R e R^m , respectivamente. Admita-se também que $z < +00$. Na sequência (P) será referido por problema primal.

O conjunto de soluções admissíveis está escrito de forma a evidenciar a distinção entre dois tipos de restrições: as explicitas, $g(x) \leqslant 0$, responsáveis sob certo ponto de vista pela dificuldade do problema, e as implícitas, que quando isoladas do primeiro grupo permitem a resolução do problema resultante de uma forma expedita. A relaxação Lagrangeana procura tirar partido desta estrutura especial.

DEFINIÇÃO

Considerando u, um vector real não negativo de dimensão m (*) a Relaxação Lagrangeana de (P) relativamente a $g(x) \leqslant 0$ é o seguinte Problema:

$$w(u) = \min_{x \in X} f(x) + ug(x) \qquad (PRu)$$

(*) se as restrições relaxadas forem da forma $g(x) \geqslant 0$, u terá então de ser não positivo; no caso $g(x)=0$, u será livre.

(PRu) é uma relaxação no sentido atrás definido pois:

1) $F(P) \subset F(PRu)$

2) $w(u) \leqslant z$

porque é válida a seguinte cadeia de relações:

$$w(u) = \min_{x \in X} f(x) + ug(x) \leqslant f(\bar{x}) + ug(\bar{x}) \leqslant f(\bar{x}) = z \quad \forall\, u>0$$

onde $\bar{x}$, é a solução óptima do problema (P). A primeira igualdade é a definição de w(u); a primeira desigualdade decorre de x ser admissível e portanto verificar $x \in X$; a segunda desigualdade deriva do produto $ug(\bar{x})$ ser não positivo ($u \geqslant 0$ por definição e $g(\bar{x}) \leqslant 0$ por $\bar{x}$ ser admissível).

A definição acima apresentada encerra uma infinidade de relaxações cada uma das quais associada a um particular vector, $u \geqslant 0$, de multiplicadores. A sequência lógica das técnicas Lagrangeanas é então a pesquisa dos multiplicadores óptimos traduzida no seguinte problema:

$$w = \max_{u \geqslant 0} w(u) \qquad (D)$$

O problema (D) designa-se por dual Lagrangeano e a sua função objectivo, w(u), por função dual.

A resolução do dual Lagrangeano, não permite, regra geral, a obtenção da solução do primal, no entanto, é fácil ver que os valores dos óptimos dos problemas (P) e (D) verificam o seguinte teorema:

TEOREMA (dualidade fraca)

$w \leqslant z$

A demonstração deste teorema decorre imediatamente de w ser o valor do óptimo de uma relaxação Lagrangeana do problema (P).

Os teoremas seguintes estabelecem as condições particulares em que o dual Lagrangeano resolve o primal, situação de maior sucesso da dualidade Lagrangeana.

==========

TEOREMA (condições suficientes de optimalidade)

Considere-se um par $(\tilde{x},\tilde{u})$, $\tilde{x} \in X$ e $\tilde{u} \geqslant 0$, satisfazendo as seguintes condições:

1- $f(\tilde{x}) + \tilde{u}g(\tilde{x}) = \min_{x \in X} f(x) + \tilde{u}g(x)$

2- $\tilde{u}g(\tilde{x}) = 0$

3- $g(\tilde{x}) \leqslant 0$

Então $\tilde{x}$ é óptimo do problema primal, $\tilde{u}$ é óptimo do problema dual e $w(\tilde{u}) = z$.

==========

Este teorema permite, em presença de um determinado par (x,u) que verifique as condições 1 a 3, afirmar que x é solução do problema primal. No entanto, por se tratar de condições suficientes, nada permitem concluir fora desta situação, nada garantindo à priori, quanto à possibilidade de através da resolução do problema dual se encontrar o óptimo do primal. Esta questão é objecto do teorema seguinte:

==========

TEOREMA (condições necessárias de optimalidade)

Se f(x) e g(x) forem funções convexas, X um convexo de R^n e se a seguinte condição suplementar for verificada

4- existe $x \in X$ tal que $g(x)<0$

Então existe $\tilde{x}$ óptimo para o primal e $\tilde{u}$ óptimo para o dual tais que o par $(\tilde{x},\tilde{u})$ satisfaz as condições 1-, 2- e 3-.

==========

O teorema anterior está intimamente relacionado com um dos resultados teóricos mais importantes da programação matemática: a equivalência entre dualização e convexificação. Uma exposição bastante clara sobre este assunto encontra-se em Shapiro(1979).

Por definição $w(u) \leqslant f(x) + ug(x)$ para todo o $x \in X$, $\forall\, u \geqslant 0$. Portanto representando os valores $[f(x),g(x)]$ em R^{m+1} (veja-se a figura) verifica-se que o hiperplano $w(u)=y_o+uy$ fica a baixo desse conjunto de pontos; substituindo $y=g(x)$ vem $y_o=w(u)-ug(x) \leqslant f(x)$. $w(u)$ é também a intersecção deste hiperplano com $y=0$.

Seja o seguinte conjunto:

$$[f,g] = \bigcup_{x \in X} \{(m,s): m \geqslant f(x) \text{ e } s \geqslant g(x)\}$$

e represente-se por $[f,g]^c$ o envolucro convexo de $[f,g]$. Seja

$$z(s) = \min \{ m: (m,s) \in [f,g]^c \}$$

que se tomará igual a $+\infty$ se não existir nenhum valor $(m,s) \in [f,g]^c$. $z^c = z^c(0)$ é o valor do problema primal convexificado.

TEOREMA

O valor do óptimo da função objectivo do dual é igual ao valor do óptimo da função objectivo do problema primal convexificado, isto é, $z^c = w$.

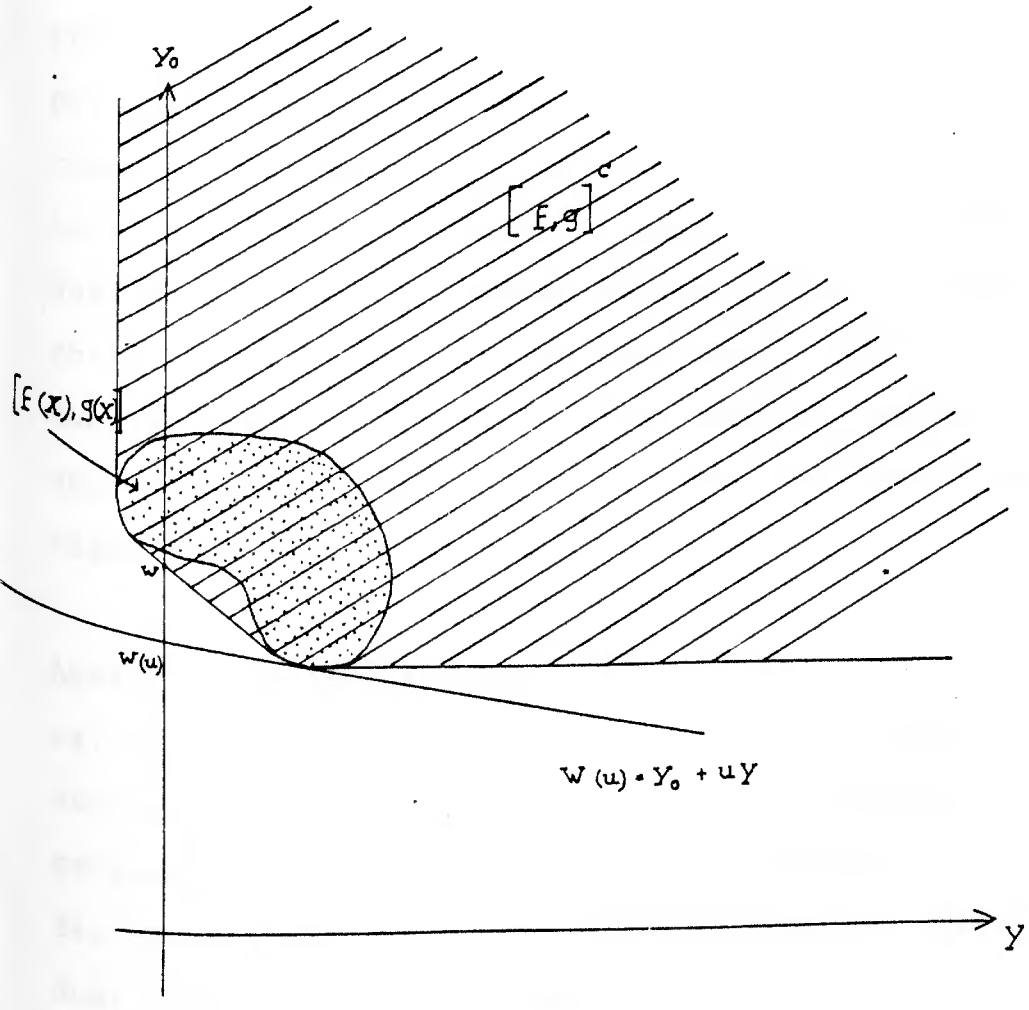


Figura 4.1

Os resultados apresentados anteriormente são extremamente interessantes do ponto de vista teórico. No entanto, o problema em estudo neste trabalho, como acontece com muitos outros de interesse prático, é um problema linear discreto e não convexo e, portanto, o pro-

cedimento da dualidade Lagrangeana anteriormente descrito conduz, com uma certa frequência, a um hiato de dualidade (duality gap), isto é, a uma diferença positiva entre os valores óptimos dos dois problemas. Bell e Shapiro (1977) demonstraram ser possível, em termos teóricos e para problemas de programação inteira, construir uma sucessão de problemas duais, cada um dos quais mais forte do que o seu predecessor, até ser encontrada uma solução óptima do primal ou provado que este é impossível. No entanto, os algoritmos concebidos por estes autores baseados na teoria dos grupos abelianos não são computacionalmente eficientes. Paulo Bárcia (1985) desenvolve uma teoria de reforço do dual e apresenta algoritmos que reforçam o dual mediante a introdução de restrições suplementares do tipo mochila binária, que permitem num número finito de iterações obter um dual que resolve o primal. Estes métodos apresentam ainda a vantagem de serem generalizáveis ao caso não linear, mediante a verificação de certas condições.

Apesar do que se afirmou no último parágrafo a dualidade Lagrangeana, sem qualquer mecanismo de reforço do dual, permanece como uma abordagem interessante, não podendo, no entanto, ser encarada duma perspectiva tão optimista. Dito de outro modo, apesar de não se poder garantir a solução do problema primal através da resolução do dual Lagrangeano, a resolução deste último continua válida como uma aproximação ao primeiro, pois esta é uma forma atractiva para determinar minorantes.É fácil concluir que os minorantes obtidos deste modo são garantidamente não inferiores aos obtidos através da omissão pura e simples das restrições explícitas, pois esta equivale à relaxação Lagrangeana com u=0, sendo portanto também considerada.

Deve referir-se que a resolução do problema dual levanta algumas dificuldades porque a função dual nem sempre é diferenciável em toda a parte, nomeadamente no caso em que X é um conjunto discreto de de pontos, como acontece com os problemas de optimização combinatória onde se inclui o problema em estudo neste trabalho. No entanto, esta função verifica propriedades que permitem a resolução do problema dual através das técnicas da optimização subgradiente parecendo ser esta a abordagem mais indicada neste tipo de situações (veja-se anexo).

A eficiência da dualidade Lagrangeana na resolução do primal depende também de elementos até aqui omitidos: a formalização concreta do problema e o modo como é subdividido o conjunto de restrições. Destes dois factores dependem quer a qualidade dos resultados obtidos quer a facilidade de resolução do problema dual. A importância deste último aspecto decorre imediatamente do facto de cada valor da função dual ser obtido através da resolução de uma relaxação do problema primal.

Estas questões serão clarificadas na secção seguinte onde se discutem duas formas de obtenção de minorantes para o PCVC que derivam de duas formalizações alternativas deste problema. Acentuar-se-à também a importância da facilidade de resolução da árvore-1 mínima, isto é, da relaxação do PCVC utilizada.

4.2 DUAS FORMAS DE DETERMINAÇÃO DE MINORANTES PARA O PCVC.

O objectivo desta secção é apresentar duas formas de determinar minorantes para o PCVC: uma utilizada no algoritmo de Jongens e Volgenant (1985), outra sugerida por estes autores tendo como ponto de partida um formalização alternativa do PCVC que conduz a um problema dual com um maior número de muliplicadores. Qualquer destas formas baseia-se na relaxação Lagrangeana do PCVC no problema da determinação da árvore-1 mínima.

O conceito de árvore-1 surge com Held e Karp (1970,1971) revelando-se uma forma expedita para a determinação de minorantes para o PCV simétrico sobretudo com as inovações do artigo de 1971.

Seja G um grafo completo e não orientado com conjunto de vértices N={1,2,...,n}. Uma árvore geradora é um subgrafo conexo sem ciclos (isto é, uma árvore) com conjunto de vértices N. O conceito de árvore-1 é uma pequena variante deste conceito.

DEFINIÇÃO

Uma árvore-1 é uma árvore no conjunto de vértices {2,...,n} à qual se juntam duas arestas, distintas, incidentes no vértice 1.

Associe-se agora a cada uma das arestas (i,j) do grafo G uma distância ou peso c_{ij} . Pode então definir-se uma árvore-1 mínima como sendo uma árvore mínima no conjunto de vértices {2,...,n} conjuntamente as duas menores arestas incidentes no vértice 1. O problema

da determinação da árvore-1 mínima é de fácil resolução pois resume-se práticamente ao problema da determinação da árvore geradora mínima que é um dos problemas completamente resolvidos da teoria dos grafos. Este assunto será retomado no final desta secção.

Nos trabalhos de Held e Karp (1970,1971) exploram-se as relações entre árvores-1 e o percurso do caixeiro viajante:

1- Um percurso é simplesmente uma árvore-1 em que cada vértice tem grau dois;
2- Se uma árvore-1 mínima é um circuito Hamiltoniano então é o circuito Hamiltoniano de peso mínimo.
3- Para qualquer vector real de dimensão n, $\pi=(\pi_1,\pi_2,\ldots,\pi_n)$ a transformação do peso das arestas

$$\hat{c}_{ij}=c_{ij}+\pi_i+\pi_j$$

não altera a solução do PCV mas modifica a árvore-1 mínima.

Quando se considera o problema do caixeiro viajante classificado, com consideração de m classes, introduz-se uma partição de cardinalidade m no conjunto de vértices. As arestas do grafo podem classificar-se em arestas locais e não locais consoante representem ligações entre vértices de uma mesma classe ou vértices de classes diferentes.

O PCVC pode então apresentar-se como sendo o problema da determinação do circuito Hamiltoniano com m arestas não locais de comprimento mínimo. As relações entre a solução do PCVC e as àrvores-1 são muito semelhantes às se estabelecem entre o PCV e as àrvores-1 havendo apenas que reformular ligeiramente as condições 1 e 2, acres-

centando-lhes a exigência de que o número de arestas não locais numa solução admissível seja igual a m, isto é, estas condições passam a ter a seguinte redacção:

1- um circuito admissível é simplesmente uma árvore-1 em que cada vértice tem grau dois e o número de arestas não locais é exactamente igual a m.

2- Se uma árvore-1 mínima é um circuito Hamiltoniano com m arestas não locais então é a solução do PCVC.

Seguidamente utilizando os resultados da secção anterior procura-se mostrar que o problema da determinação da árvore-1 mínima com a seguinte transformação nos pesos das arestas:

$$\hat{c}_{ij} = \begin{cases} c_{ij} + \pi_i + \pi_j & \text{se } (i,j) \text{ é uma aresta local} \\ c_{ij} + \pi_i + \pi_j + \mu & \text{se } (i,j) \text{ é uma aresta não local} \end{cases} \quad (4.1)$$

com π vector real de dimensão n e μ real é uma relaxação Lagrangeana do PCVC. Esta relaxação foi utilizada por Jongens e Volgenant (1985) para determinar minorantes para o PCVC.

Com esse objectivo vai apresentar-se uma formalização do PCVC como problema de PLI apenas válida para a versão simétrica.

Considere-se, como no capítulo anterior, que:

$N=\{1,2,\ldots,n\}$ é o conjunto de n cidades;

$M=\{1,2,\ldots m\}$ é o conjunto de m classes;

G_k é o conjunto das cidades que formam a classe k;

x_{ij} é uma variável que assume o valor 1 se a ligação (i,j) está na solução e 0 caso contrário;

$C=[c_{ij}]$ uma matriz simétrica cujo elemento genérico representa a distância da ligacão directa da cidade i à cidade j. As cidades foram renumeradas, se necessário, para ser possível a seguinte decomposição da matriz C:

$$\begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ & \cdots & & \\ C_{m1} & C_{m2} & \cdots & C_{mm} \end{bmatrix}$$

onde os elementos dos blocos $C_{KK'}$ $k=1,2,\ldots,m$, representam distâncias entre cidades da classe k e os elementos de C_{lK} distâncias entre cidades da classe k e cidades da classe l com $l \neq k$.

O problema do caixeiro viajante classificado e simétrico pode ser formalizado como o seguinte programa de PLI:

$$\min \sum_{1 \leqslant i < j \leqslant n} c_{ij} \; x_{ij} \qquad (4.2)$$

sa

$$\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji} = 2 \qquad i=1,2,\ldots,n \qquad (4.3)$$

$$\sum_{\substack{i \in S \\ j \in S \\ i<j}} x_{ij} \leqslant |S|-1 \qquad \text{para } \forall\, S \subset \{2,3,\ldots n\} \text{ com } |S| \geqslant 1 \qquad (4.4)$$

$$\sum_{k=1}^{m} \sum_{\substack{i \in G_K \\ j \in N-G_K \\ i<j}} x_{ij} = m \qquad (4.5)$$

$$x_{ij} \in \{0,1\} \qquad 1 \leqslant i < j \leqslant n \qquad (4.6)$$

Held e Karp (1970) demonstraram que o conjunto de restrições (4.3) pode ser substituido pelo seguinte:

$$\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji} = 2 \qquad i=1,2,\ldots,n-1 \qquad (4.7)$$

$$\sum_{1\leqslant i<j\leqslant n} x_{ij} = n \qquad (4.8)$$

Pode portanto rescrever-se o programa:

$$\min \sum_{1\leqslant i<j\leqslant n} c_{ij}\ x_{ij} \qquad (4.9)$$

sa

$$\sum_{i<j} x_{ij} + \sum_{j<i} x_{ji} = 2 \qquad i=2,\ldots,n-1 \qquad (4.10)$$

$$\sum_{j} x_{1j} = 2 \qquad (4.11)$$

$$\sum_{1\leqslant i<j\leqslant n} x_{ij} = n \qquad (4.12)$$

$$\sum_{\substack{i\in S\\ j\in S\\ i<j}} x_{ij} \leqslant |S|-1 \qquad \text{para } \forall\ SC\{2,3,\ldots n\} \text{ com } |S|>1 \qquad (4.13)$$

$$\sum_{k=1}^{m} \sum_{\substack{i\in G_K\\ j\in N-G_K\\ i<j}} x_{ij} = m \qquad (4.14)$$

$$x_{ij} \in \{0,1\} \qquad 1\leqslant i<j\leqslant n \qquad (4.15)$$

Se a este problema forem retiradas as restrições (4.10) e (4.14) o problema resultante é o da determinação da árvore-1 mínima.

De acordo com o exposto na secção anterior, para se obter a partir deste último problema uma relaxação Lagrangeana do PCVC, devem ponderar-se as restrições relaxadas com multiplicadores (*) e transportá-las para a função objectivo de modo a penalizar a sua violação.

Considere-se então um vector π real de dimensão n para ponderar as restrições (4.10) e μ real ponderador da restrição (4.14), a função objectivo (4.9) de acordo com o que anteriormente se disse pode escrever-se:

$$w(\pi,\mu) = \min \sum_{1\leqslant i<j\leqslant n} c_{ij}\, x_{ij} + \sum_{j=2}^{n-1} x_{1j}\,\pi_j + \sum_{i=2}^{n-1} x_{in}\,\pi_i + \qquad (4.16)$$

$$+ \sum_{i=2}^{n-2} \sum_{j=i+1}^{n-1} x_{ij}\,(\pi_i+\pi_j) - 2\sum_{i=2}^{n-1}\pi_i + \sum_{k=1}^{m} \sum_{\substack{i\in G_K \\ j\in N-G_K \\ i<j}} \mu x_{ij} - m\mu$$

Fazendo $\pi_1 = \pi_n = 0$

(*) neste caso livres porque as restrições relaxadas são de igualdade.

$$w(\pi,\mu)= \min \sum_{1\leqslant i<j\leqslant n} c_{ij}\, x_{ij} + \sum_{j=2}^{n-1} x_{1j}\,\pi_j + \sum_{j=2}^{n} x_{1j}\,\pi_1 + \sum_{i=2}^{n-1} x_{in}\,\pi_i +$$

(4.17)

$$+ \sum_{i=1}^{n-1} x_{in}\,\pi_n + \sum_{i=2}^{n-2}\sum_{j=i+1}^{n-1} x_{ij}\,(\pi_i+\pi_j) - 2\sum_{i=1}^{n}\pi_i + \sum_{k=1}^{m} \sum_{\substack{i\in G_K \\ j\in N-G_K \\ i<j}} \mu x_{ij} - m\mu$$

ou ainda,

$$w(\pi,\mu)=\min \sum_{1\leqslant i<j\leqslant n} c_{ij}\, x_{ij} + \sum_{1\leqslant i<j\leqslant n} x_{ij}\,(\pi_i+\pi_j) + \sum_{k=1}^{m} \sum_{\substack{i\in G_K \\ j\in N-G_K \\ i<j}} \mu x_{ij} +$$

(4.18)

$$-2\sum_{i=1}^{n}\pi_i - m\mu$$

decompondo os dois primeiros somatórios em arestas locais e não locais e reagrupando pode escrever-se

$$w(\pi,\mu)=\min \sum_{k=1}^{m} \sum_{\substack{i\in G_K \\ j\in G_K \\ i<j}} (c_{ij}+\pi_i+\pi_j)x_{ij} + \sum_{k=1}^{m} \sum_{\substack{i\in G_K \\ j\in N-G_K \\ i<j}} (c_{ij}+\pi_i+\pi_j+\mu)x_{ij} - R$$

(4.19)

$$\text{com } R = 2\sum_{i=1}^{n}\pi_i + m\mu$$

Esta expressão é precisamente a expressão da função objectivo do problema da determinação da árvore-1 mínima com as distâncias de acordo com a transformação (4.1).

Fica então demonstrado que relaxando Lagrangeanamente as restrições (4.10) e (4.14) se obtem o problema da determinação da árvore-1 mínima com uma matriz de custos transformada de acordo com (4.1).

Deste modo obteve-se uma família de problemas, cada um dos quais associados a um particular conjunto de valores $(\pi_1, \pi_2, \ldots, \pi_n, \mu)$ cujos valores óptimos são minorantes para o valor óptimo do PCVC. Trata-se agora de seleccionar o problema que permite obter o maior desses minorantes. De acordo com os resultados da secção anterior esta escolha faz-se resolvendo o problema dual Lagrangeano associado, isto é, determinando

$$W = \underset{\pi,\mu}{\text{Max}}\ w(\pi,\mu) \qquad (4.20)$$

Jongens e Volgenant utilizam este método no seu algoritmo mas sugerem a seguinte alternativa de transformação nas distâncias:

$$\hat{c}_{ij} = \begin{cases} c_{ij} + \pi_i + \pi_j & \text{se } i,j \in G_K \\ c_{ij} + \pi_i + \pi_j + \mu_l + \mu_K & \text{se } i \in G_l \text{ e } j \in G_K \text{ com } k \neq l \end{cases} \qquad (4.21)$$

como uma possibilidade de obter minorantes que eventualmente conduza a uma aceleração do método de partições e avaliações sucessivas por eles proposto para a resolução do PCVC simétrico.

O problema da determinação da árvore-1 mínima com a transformação da matriz das distâncias de acordo com (4.21) pode também ser obtida como uma relaxação Lagrangeana do PCVC.

Esta relaxação tem como ponto de partida uma formalização do PCVC equivalente a (4.9) a (4.15) onde a restrição (4.14) é substituida

pelo conjunto:

$$\sum_{\substack{i\in G_K \\ j\in N-G_K \\ i<j}} x_{ij} + \sum_{\substack{i\in G_K \\ j\in N-G_K \\ j<i}} x_{ji} = 2 \qquad k=1,2,\ldots,m \qquad (4.22)$$

que traduz directamente a exigência de que o número de arestas (não locais) incidentes em cada classe seja exactamente 2, isto é, traduz de uma forma alternativa a (4.14) a obrigatoriedade de visita contínua às cidades de cada classe.

Seguindo um processo análogo ao anterior e associando ás restrições (4.22) o vector de multiplicadores $\hat{\mu}$ (real de dimensão m) obtem-se para expressão da função objectivo do problema relaxado

$$\hat{w}(\pi,\hat{\mu}) = \sum_{k=1}^{m} \sum_{\substack{i\in G_K \\ j\in G_K \\ i<j}} (c_{ij} + \pi_i + \pi_j) x_{ij} + \sum_{1\leqslant l<k\leqslant m} \sum_{\substack{i\in G_l \\ j\in G_K \\ i<j}} (c_{ij} + \pi_i + \pi_j + \hat{\mu}_l + \hat{\mu}_K) x_{ij} - \hat{R} \qquad (4.23)$$

com $\hat{R} = 2\sum_{k=1}^{m} \hat{\mu}_K + 2 \sum_{i=1}^{n} \pi_i$.

O melhor minorante associado a esta nova família de relaxações, obtem-se resolvendo o dual Lagrangeano associado, isto é, determinando

$$\hat{W} = \max_{\pi,\hat{\mu}} \hat{w}(\pi,\hat{\mu}). \qquad (4.24)$$

Dispõe-se então de duas formas alternativas (4.20) e (4.24) para a determinação de minorantes para o valor óptimo do PCVC, ambas baseadas em relaxações no problema da determinação da árvore-1 mínima. Quer isto dizer que a obtenção de cada um dos valores das funções duais (isto é, das funções definidas em (4.19) e (4.23)) exige a resolução de um destes problemas. Ou seja, a facilidade de determinação destes minorantes depende em grande parte da facilidade de resolução do problema da determinação da árvore-1 mínima.

Para finalizar esta secção far-se-á então uma breve referência aos métodos de resolução deste último problema.

DETERMINAÇÃO DA ÁRVORE-1 MÍNIMA

O problema da determinação da árvore-1 mínima pode ser decomposto nos seguintes:

- selecção das duas menores arestas incidentes no vértice 1
- determinação de uma árvore geradora mínima no conjunto de vértices N-{1}.

O primeiro destes problemas é trivial pois resume-se à escolha das duas menores arestas num conjunto de n.

Para a resolução do segundo problema existem dois métodos clássicos. Um deve-se a Kruskal (1956) e consiste em examinar, uma a uma todas as arestas do grafo por ordem decrescente do seu peso. Se uma aresta, e , em exame não forma ciclo (quando adicionada às arestas já seleccionadas) é incluida na árvore mínima, T, caso contrário é i-

gnorada. O procedimento pára quando já estão encontradas n-2 arestas ou quando já se examinaram todas as arestas do grafo. Se o grafo é desconexo o resultado do algoritmo é uma floresta geradora mínima. Concretamente o algoritmo de Kruskal para determinar a árvore mínima no conjunto de vértices N-{1} pode ser descrito do seguinte modo:

```
T := ∅
E :={conjunto das arestas do subgrafo}
WHILE |T| < n-2 AND E <> ∅ DO
 BEGIN
   e := menor aresta de E
   E := E - {e}
   if { T U {e}} não contém ciclos then T := T U {e}
 END
IF |T| < n-2 THEN WRITE(grafo desconexo)
```

O outro algoritmo clássico para a resolução deste problema deve-se a Prim (1957) ou Dijkstra (1959) e consiste em partindo de um vértice arbitrário, s, procurar a aresta de menor peso incidente em s, (s,t) sendo esta a primeira aresta da árvore, T. Seguidamente vão sendo incluidas as restantes n-3 arestas em T, sendo em cada passo incluida a menor das arestas que liga um dos vértices de V_t, já considerados em T, a um dos vértices de $V-V_t$. Esquemáticamente:

```
escolher arbitrámente um vértice s
V_t := {s}
E_t := {(i,s): i∈ N-{s}}
T := ∅
```

```
WHILE |Vt| < n-2
 BEGIN
   (i,j) menor aresta de Et
   Vt := Vt U {j}
   T := T U {(i,j)}
   Et := Et - {(k,j): k ∈ Vt} U {(j,t): t ∈ V-Vt}
 END
IF |Vt| < n-2 THEN WRITE(grafo desconexo)
```

Estes dois métodos apesar de conduzirem ao mesmo resultado regra geral não seleccionam as arestas pela mesma sequência. Não sendo possível afirmar categoricamente qual dos algoritmos é mais rápido parece no entanto aconselhável a utilização do algoritmo de Prim em grafos de pequena dimensão sendo o algoritmo de Kruskal mais indicado para grafos pouco densos, Syslo et al. (1983).

Com o objectivo de comparar as duas alternativas apresentadas para determinação de minorantes para o valor óptimo do PCVC foram utilizados os problemas do capítulo anterior. Uma vez que estes problemas correspondem a grafos completos a determinação das árvores-1 mínimas fez-se com base no método de Prim. Na secção seguinte apresentam-se os resultados desse trabalho computacional.

4.3 RESULTADOS COMPUTACIONAIS

Nesta secção analisam-se os resultados de testes efectuados com o objectivo de comparar as duas formas alternativas para a determinação de minorantes para o valor óptimo do PCVC simétrico apresentadas na secção anterior. No seguimento designar-se-à por M1 a pri-

meira forma determinação de minorantes apresentada, isto é, a que resulta do problema (4.20), e por M2 a segunda forma, ou seja, a que deriva do problema (4.24).

Para a realização dos testes foi utilizado o mesmo conjunto de 73 problemas que serviu para comparar os métodos heurísticos na última secção do capítulo anterior. Como se referiu, neste conjunto incluem-se problemas com dimensões (isto é, número de cidades e número de classes) e estruturas de dados (DT0, DT1 e DT2) diversas, permitindo-se deste modo uma análise do efeito destes aspectos no comportamento dos métodos em estudo. Os detalhes sobre este conjunto de problemas podem ser encontrados em anexo.

Como qualquer um dos minorantes é obtido através da resolução de um problema dual, foi elaborado um programa em Pascal para resolver estes problemas pelo método subgradiente com a seguinte estrutura:

```
(* inicialização *)
  k:=0
  kact:=0
  u_o:=0
  determinar uma árvore-1 com u_o
  solrel:={arestas da árvore-1}
  determinar g_o ∈ Dw(u_o)
  melhormin:=valor do minorante obtido com u_o
  IF ||g_o||=0 THEN BEGIN
                      parar:=true
                      WRITE(foi obtida uma solução do primal)
                    END
               ELSE parar:=false
```

```
(* iteração *)
  WHILE (k<400) AND (NOT parar) DO
    BEGIN
      u_{K+1} :=u_K+passo*g_K
      determinar árvore-1 com u_{K+1}
      determinar g_{K+1} ∈ Dw(u_{K+1})
      min:=valor do minorante obtido com u_{K+1}
      IF min>melhormin THEN
                        BEGIN
                          melhormin:=min
                          solrel:={arestas da árvore-1 com u_{K+1}}
                          kact:=0
                        END
                       ELSE kact:=kact+1
      IF kact>50 THEN parar:=true
      IF ||g_{K+1}||=0 THEN
                      BEGIN
                        parar:=true
                        WRITE(foi obtida uma solução do primal)
                      END
      K:=K+1
    END
```

O passo do método foi determinado utilizando 3 estratégias alternativas:

E1- passo constante e igual a 1 como Held e Karp (1971);

E2- passo= $p_K \dfrac{\bar{w} - w(u_K)}{||g_K||^2}$ onde $\bar{w}$ é um majorante de w e p_K começa por tomar o valor 2 sendo dividido por 2 sempre que em 5 iterações consecutivas não se consegue melho-

rar o valor do minorante;

E3- muito semelhante à estratégia anterior e conforme sugestão de Held et al. (1974). Foram, no entanto, introduzidas ligeiras alterações resultantes de algumas experiências realizadas. Concretamente, o passo é determinado por uma expressão idêntica à anterior, mas agora p_K começa por tomar o valor 1 que se mantem durante as primeiras n/2 iterações, seguidamente p_K e o número de iterações que é mantido constante são ambos divididos por dois até que o último atinja o valor 5 a partir do qual p_K é dividido por dois de 5 em 5 iterações.

Em cada problema o majorante, $\bar{w}$, requerido pela estratégias E2 e E3 foi o menor dos majorantes obtidos com os métodos heurísticos do capítulo anterior. As árvores-1 mínimas foram determinadas com base no algoritmo de Prim, pois os problemas que serviram para efectuar os testes correspondem a grafos completos. Com este fim foi utilizado um programa em Pascal apresentado por Syslo et al. (1983) com ligeiras alterações.

Os resultados em seguida apresentados decorrem da aplicação do método subgradiente a cada uma das alternativas em estudo, M1 e M2, com as três estratégias de determinação do passo acima referidas, E1, E2 e E3. Deste modo para cada problema obtêm-se 6 minorantes, cada um dos quais resulta da aplicação de um método alternativo, isto é, de uma combinação de uma estratégia para a determinação do passo no método subgradiente com uma das formas de obtenção de minorantes.

Tal como a comparação dos métodos heurísticos, os métodos em estudo serão analisadas tendo em atenção dois aspectos essenciais: a qualidade dos valores obtidos, neste caso dos minorantes, e a rapidez com que são determinados.

Quanto ao primeiro aspecto os métodos serão também comparados mediante o número de vezes que cada um obteve o minorante de valor mais alto e ainda pela média dos desvios percentuais relativamente ao melhor valor obtido para cada problema. A rapidez dos diversos métodos será avaliada pelo tempo de CPU gasto na sua execução.

Quanto à comparação dos métodos relativamente à qualidade dos minorantes que determinam, analisem-se em primeiro lugar os quadros seguintes cujos valores referem o número de vezes que cada um obteve o minorante de valor mais alto, e nos quais problemas de igual dimensão se encontram agrupados.

PRIMEIRA FORMA DE DETERMINAÇÃO DE MINORANTES

nº de cidades	nº classes	M1E1	M1E2	M1E3
24	7	1	1	1
80	6	0	2	1
	12	0	1	1
100	8	0	1	1
	15	0	2	1
120	9	0	1	0
	18	0	2	1
150	11	0	1	0
	22	0	3	0
TOTAL (*)		1	14	6

SEGUNDA FORMA DE DETERMINAÇÃO DE MINORANTES

nº de cidades	nº classes	M2E1	M2E2	M2E3
24	7	1	1	1
80	6	1	6	4
	12	2	5	3
100	8	0	6	5
	15	0	8	3
120	9	0	5	3
	18	1	5	3
150	11	0	5	4
	22	0	4	3
TOTAL (*)		5	45	29

(*) a soma dos totais é diferente de 73 porque para alguns problemas o melhor minorante foi obtido simultânemente por mais do que uma alternativa.

Da análise destes valores sobressai uma superioridade da segunda forma de determinação de minorantes generalizável a todas as estratégias para o passo do método subgradiente. Das 6 alternativas em estudo a combinação que, sob este ponto de vista, permite a obtenção de melhores resultados é a da segunda forma, M2, com a estratégia E2, seguindo-se-lhe não muito afastada a combinação M2E3. Um aspecto curioso a salientar é a insensibilidade dos métodos, sob este ponto de vista, relativamente à dimensão dos problemas.

Os quadros em baixo referem-se à mesma informação, isto é, ao número de vezes que cada método encontrou o melhor minorante, mas agora agrupam-se os problemas por tipo de dados.

PRIMEIRA FORMA DE DETERMINAÇÃO DE MINORANTES

tipo de dados	M1E1	M1E2	M1E3
DT0	1	6	2
DT1	0	8	4
DT2	0	0	0

SEGUNDA FORMA DE DETERMINAÇÃO DE MINORANTES

tipo de dados	M2E1	M2E2	M2E3
DT0	1	21	5
DT1	1	21	5
DT2	3	3	19

Estes quadros permitem confirmar a supremacia da segunda forma de determinação de minorantes mas esclarecem ainda que a estratégia para o passo do método subgradiente E2 é claramente superior às outras estratégias apenas nos dados dos tipos DT0 e DT1, sendo a estratégia E3 preferível em presença de problemas que representam configurações espaciais onde as cidades de cada classe estão próximas como é o caso dos dados do tipo DT2.

Os métodos em estudo podem ainda ser comparados sob este aspecto analisando-se os quadros seguintes que apresentam a média dos desvios percentuais relativamente ao maior valor obtido.Nestes quadros a informação está organizada por dimensão de problema.

PRIMEIRA FORMA DE DETERMINAÇÃO DE MINORANTES

nº de cidades	nº classes	M1E1	M1E2	M1E3
24	7	0	0	0
80	6	4.569	4.275	3.999
	12	3.098	2.008	2.126
100	8	3.422	2.372	1.679
	15	5.932	2.703	2.681
120	9	3.550	2.560	1.607
	18	1.293	0.543	0.276
150	11	1.632	1.002	1.176
	22	1.132	0.396	0.583
Média		3.036	1.955	1.742

SEGUNDA FORMA DE DETERMINAÇÃO DE MINORANTES

nº de cidades	nº classes	M2E1	M2E2	M2E3
24	7	0	0	0
80	6	0.289	0.260	0.454
	12	0.240	0.256	0.307
100	8	0.677	0.213	0.012
	15	3.856	3.333	5.018
120	9	1.630	1.001	0.018
	18	0.473	0.092	0.465
150	11	0.508	0.202	0.719
	22	0.537	0.253	0.018
Média		1.012	0.691	0.864

O elemento novo que ressalta da análise destes quadros é a ausência de grandes contrastes entre os valores obtidos pelos diversos métodos. Verifica-se mais uma vez uma insensibilidade dos métodos à di-

mensão dos problemas e uma hierarquização semelhante das alternativas em estudo.

Os quadros seguintes referem-se à mesma informação mas agora organizada por tipos de dados.

PRIMEIRA FORMA DE DETERMINAÇÃO DE MINORANTES

	M1E1	M1E2	M1E3
DT0	0.237	0.030	0.184
DT1	1.280	0.684	0.275
DT2	7.708	5.233	4.832
Média	3.036	1.955	1.742

SEGUNDA FORMA DE DETERMINAÇÃO DE MINORANTES

	M2E1	M2E2	M2E3
DT0	0.281	0.003	0.034
DT1	0.851	0.583	1.032
DT2	1.935	1.518	1.561
Média	1.012	0.691	0.864

Estes valores permitem concluir que é nos dados do tipo DT2 que se verificam vantagens mais evidentes da segunda forma de determinação de minorantes.

Nos quadros seguintes encontram-se os tempos de CPU gastos na execução de cada um dos métodos num VAX 780, medidos em segundos, encontrado-se a informação organizada por problemas de igual dimensão.

PRIMEIRA FORMA DE DETERMINAÇÃO DE MINORANTES

nº de cidades	nº classes	M1E1	M1E2	M1E3
24	7	34.990	4.89	3.82
80	6	249.272	223.013	136.466
	12	229.937	226.776	137.890
100	8	355.251	347.557	233.166
	15	386.020	351.151	233.842
120	9	589.454	519.559	365.742
	18	615.471	452.157	342.912
150	11	880.922	732.442	590.622
	22	843.508	753.346	580.343
MEDIA		512.102	446.642	323.187

SEGUNDA FORMA DE DETERMINAÇÃO DE MINORANTES

nº de cidades	nº classes	M2E1	M2E2	M2E3
24	7	18.92	4.99	3.46
80	6	248.868	203.122	135.344
	12	233.646	223.374	140.609
100	8	353.768	325.176	236.490
	15	366.441	321.237	236.492
120	9	602.330	525.202	368.731
	18	577.008	429.482	343.524
150	11	875.989	710.737	599.450
	22	885.742	768.743	647.404
MÉDIA		511.138	432.447	333.883

Estes quadros permitem constatar que, como seria de esperar, em termos genéricos quanto maior o número de cidades mais tempo é necessário para a determinação dos minorantes. No entanto, o número

de classes não parece afectar a rapidez dos métodos. Quanto à comparação das alternativas, a que obtem os valores mais rápidamente é a estratégia para o passo que se designou por E3. As duas formas de determinação de minorantes não se distinguem quanto a este aspecto.

A análise dos mesmos valores mas agora organizados por tipo de dados pode ser efectuda a partir dos quadros seguintes.

PRIMEIRA FORMA DE DETERMINAÇÃO DE MINORANTES

	M1E1	M1E2	M1E3
DT0	405.032	398.641	308.892
DT1	558.260	390.690	318.722
DT2	577.473	546.511	342.544
MÉDIA	512.102	446.642	323.187

SEGUNDA FORMA DE DETERMINAÇÃO DE MINORANTES

	M2E1	M2E2	M3E3
DT0	333.706	403.506	331.722
DT1	598.105	393.737	317.670
DT2	608.248	501.305	352.446
MÉDIA	511.138	432.447	333.883

Os métodos em estudo não se distinguem sensivelmente sob este aspecto. No entanto pode verificar-se que os problemas que genéricamente requerem mais tempo são os dos dados do tipo DT2.

A generalidade das alternativas apresenta tempos de computação elevados, que poderão ser reduzidos alterando ou introduzindo novos critérios de paragem. Podem alterar-se quer o número máximo de ite-

rações permitido (400),quer o número de iterações consecutivas (50) ao fim do qual se para se não houver actualização do minorante. Pode ainda introduzir-se um critério adicional de paragem estabelecendo um limite inferior ao parâmetro p_K (auxiliar do passo) nas estratégias E2 e E3.

A análise destes resultados permite concluir que a sugestão apresentada por Jongens e Volgenant (1985), isto é, a forma de determinação de minorantes M2, permite obter minorantes de valor mais elevado do que M1 e que os tempos de computação requeridos pelas duas alternativas não se distinguem.

AVALIAÇÃO DO CONJUNTO DE RESULTADOS

Até agora os resultados dos testes efectuados foram analisados separadamente. Por um lado, na última secção do capítulo anterior, compararam-se os métodos heurísticos, por outro lado, nesta secção compararam-se os métodos de determinação de minorantes. Uma análise interessante e só agora possível resulta da reunião de todos os resultados obtidos.

A qualidade das soluções aproximadas determinadas pelos métodos heurísticos pode então ser avaliada mediante uma comparação com os minorantes agora determinados. Além disso, como os valores das soluções aproximadas são majorantes do óptimo obteve-se para cada problema um intervalo que o contém. Uma forma interessante de abordar a qualidade do conjunto dos métodos em estudo neste trabalho consiste então na observação da amplitude destes intervalos, isto

é, da diferença entre o menor dos majorantes e o maior dos minorantes.

Como em alguns casos a resolução dos problemas duais permitiu obter a solução do primal pode ainda, nestes casos, avaliar-se a qualidade das soluções aproximadas fornecidas pelos métodos heurísticos comparando o seu o valor com o óptimo. Os quadros seguintes apresentam esta informação, organizando-se os resultados por dimensão de problema e por tipo de dados.

nº cidades	nº classes	Hiato % do maj (1)	nº óptimos (2)	Hiato % do opt (3)
24	7	0	1	6.699
80	6	5.811	2	10.327
	12	6.822	0	-
100	8	8.611	1	8.134
	15	6.622	0	-
120	9	9.811	0	-
	18	7.044	1	9.681
150	11	9.033	1	6.578
	22	9.044	0	-
	Média	7.742	-	8.624

TIPO DE DADOS	Hiato % do maj (1)	nº óptimos (2)	Hiato % do opt (3)
DT0	7.260	3	7.812
DT1	7.342	3	9.437
DT2	8.646	0	-
Média	7.742	-	8.624

Na coluna (1) encontram-se as médias das percentagens das amplitudes dos intervalos relativamente ao melhor majorante, isto é, a média de

$$\frac{\text{(menor majorante - maior minorante)*100.}}{\text{menor majorante}}$$

Os valores da coluna (2) referem-se ao número de problemas para os quais se determinou a solução óptima. Na coluna (3) a média das seguintes percentagens,

$$\frac{\text{(menor majorante - óptimo)*100}}{\text{óptimo}} ,$$

para o número de problemas referido na coluna (2).

Destes quadros infere-se que a dimensão dos problemas afecta a qualidade dos resultados aumentando, regra geral, a amplitude dos intervalos com o número de cidades dos problemas. No entanto, o aumento da proporção número de cidades/número de classes nem sempre actua no mesmo sentido. Quanto à análise destes valores mas por tipo de dados são os problemas que representam uma estrutura espacial com configuração de classes (DT2) nos quais se registam as amplitudes mais elevadas.

Para o conjunto de 73 problemas testados foram determinados intervalos que contêm o óptimo cuja amplitude é em média 7.742 % do menor majorante. A diferença média entre o majorante e o valor óptimo para os problemas em que este foi determinado pelo método subgradiente não é substancialmente diferente da amplitude da generalidade dos intervalos havendo assim fortes razões para suspeitar que o hiato de dualidade é reduzido. Por este motivo pode sugerir-se a introdução destes métodos para a determinação de minorantes em mé-

todos de partições e avaliações sucessivas para a resolução do PCVC, pois a dimensão da àrvore de pesquisa destes métodos depende sobretudo da qualidade dos minorantes determinados. No entanto esta utilização requer alguma atenção no sentido de reduzir os tempos de computação, como já foi referido.

5.CONCLUSÕES

Muitos problemas reais podem ser formalizados como problemas de caixeiro viajante com restrições adicionais. Quando as restrições adicionais se referem à obrigatoriedade de visitar contínuamente subconjuntos de cidades a formalização correspondente é o Problema do Caixeiro Viajante Classificado.

No Problema Classificado as cidades encontram-se agrupadas em classes exigindo-se que o caixeiro visite as cidades de cada uma das classes continuamente. O Problema do Caixeiro Viajante Classificado resulta assim da imposição de restrições de classe ao Problema do Caixeiro Viajante. Deste modo o conjunto de soluções admissíveis do PCVC é mais reduzido do que o conjunto de soluções admisíveis de um PCV com o mesmo número de cidades sendo portanto de esperar que seja possível determinar mais rapidamente a solução óptima do problema classificado. Contudo esta redução parece insuficiente para evitar um arrastamento ao problema classificado da dificuldade de resolução característica do PCV. Não se conhece, no entanto, nenhuma referência a este propósito.

Para abordar problemas deste tipo é preciso ponderar previamente as vantagens do conhecimento da solução óptima com a impossibilidade de controlo dos gastos em tempo computacional que uma tal pesquisa comporta. Assim, para resolver situações reais, quando os problemas são de dimensões razoáveis, é frequente optar pela implementação de soluções aproximadas resultantes da aplicação de métodos heurísticos, que sacrificando a garantia de obtenção do óptimo per-

mitem o conhecimento de soluções aproximadas em tempo útil. Neste trabalho desenvolveram-se 3 novos métodos heurísticos para o PCVC.

Todos os métodos heurísticos apresentados exploram de uma forma ou de outra relações entre este problema e o Problema do Caixeiro Viajante recorrendo a heurísticas para este último. O método de Jongens e Volgenant (1985), MJV, começa precisamente com a determinação de um circuito Hamiltoniano no qual à posteriori é forçada a verificação das restrições de classe. O MVPC é uma adaptação do conhecido método do vizinho mais próximo para o PCV onde se alteram ligeiramente algumas regras de modo a que o resultado final seja um circuito Hamiltoniano admissível para o PCVC. O MCICL e o MCLCI procuram tirar partido da estrutura hierárquica do PCVC recorrendo a heurísticas para o PCV na determinação dos circuitos e caminhos Hamiltonianos de ambos os níveis.

Os testes computacionais realizados com o objectivo de comparar estes métodos parecem indicar que o MVPC é de entre eles o que conduz a melhores soluções e que os mais rápidos são o MCLCI e o MCICL. No entanto, o tempo de execução dos diversos métodos foi certamente condicionado pela opção de sistematicamente efectuar um elevado número de corridas das heurísticas para o PCV, Método do Vizinho mais Próximo e Método da Inserção mais Afastada, para daí extrair em cada caso a melhor solução. Possivelmente o método que foi mais penalizado devido a esta opção foi o MVJ (pois a primeira fase implicou n corridas do MVP e n corridas do MIA) seguindo-se-lhe o MVPC cujos resultados se referem a n corridas. A uma diminuição do número de execuções corresponderá certamente uma deterioração do valor das soluções obtidas. Um trabalho interessante a realizar futuramen-

te consiste em estudar os efeitos destas duas tendências procurando, numa óptica de custo benefício, determinar um número equilibrado de corridas.

Os métodos heurísticos para o PCV foram seleccionados por apresentarem características que se julgaram adequadas à utilização que delas se pretendia fazer. No entanto, experimentação de outras heurísticas para o PCV pode conduzir a alternativas mais atraentes para a resolução dos diversos passos dos métodos para o PCVC apresentados, quer melhorando a qualidade das soluções obtidas quer diminuindo os tempos de execução.

Qualquer dos métodos heurísticos apresentado neste trabalho é do tipo construtivo. Um trabalho interessante a realizar é a aplicação de um método melhorativo, à semelhança do que foi proposto por Lin (1965) para o problema do Caixeiro Viajante, que utilize como ponto de partida as soluções determinadas pelos diversos métodos.

No capítulo 4 deste trabalho estudaram-se ainda duas formas alternativas de obtenção de minorantes para o valor óptimo do PCVC sugeridas por Jongens e Volgenant (1985). Qualquer destas alternativas tem como suporte uma adaptação ao problema classificado da relaxação no problema da determinação da àrvore-1 mínima utilizada por Held e Karp (1970,1971) na resolução do problema do caixeiro viajante. A função objectivo desta relaxação é reforçada pela introdução de um ou de m multiplicadores adicionais para penalizar a violação das restrições de classe. Os testes computacionais efectuados parecem indicar que a utilização de m multiplicadores adicionais faz aumentar, ainda que de forma não muito significativa, o valor

dos minorantes sem implicar um aumento do tempo de computação necessário à sua determinação. A justificação da superioridade desta alternativa pode talvez encontrar-se na maior flexibilidade de penalização das restrições de classe. Enquanto a utilização de m multiplicadores adicionais permite alterar a atracção relativa das classes, possibilitando um ajustamento idêntico ao que se realiza ao nível das cidades, a consideração de apenas um multiplicador adicional impõe o tratamento de todas as classes de igual modo agravando ou aliviando o peso da generalidade das arestas não locais.

Uma sugestão para trabalho futuro consiste em estudar regras para tornar admissíveis para o PCVC as soluções dos problemas relaxados e, deste modo, obter eventualmente boas soluções aproximadas.

Da conjugação dos resultados computacionais dos capítulos 3 e 4 é possível obter intervalos que contêm o óptimo de cada um dos problemas testados. Estes intervalos com uma amplitude média de 7.742% (do menor majorante determinado em cada caso) sugerem que as soluções heurísticas obtidas pelo conjunto de métodos desenvolvidos constituem aproximações razoáveis do valor óptimo o que recomenda a sua utilização.

Para 6 dos problemas testados algumas das alternativas de determinação de minorantes conduziram à obtenção de soluções admissíveis e, portanto óptimas. Para estes problemas foi então determinado exactamente o erro dos majorantes. Como estes erros não são significativamente diferentes dos erros por excesso obtidos para os restantes problemas testados, é lógico admitir-se que os processos de

determinação de minorantes conduzem a hiatos de dualidade reduzidos.

Para resolver problemas difíceis são com frequência utilizados métodos de partições e avaliações sucessivas, sendo a dimensão das árvores de pesquisa destes métodos em grande medida função do valor dos minorantes determinados. Assim sendo, pode recomendar-se a incorporação dos processos de determinação de minorantes estudados neste trabalho em métodos de partições e avaliações sucessivas para a resolução do PCVC. Contudo, para este tipo de utilização talvez seja vantajoso realizar algum trabalho no sentido de reduzir os tempos de computação destes métodos. Com este objectivo pode proceder-se à semelhança do proposto por Held e Karp (1971) não insistindo demasiado na optimização subgradiente pois, como se sabe estes métodos conduzem rapidamente a soluções bastante razoáveis e, para o seu refinamento gastam um elevado número de iterações sem alterações sensíveis.

ANEXO 1 - MÉTODO SUBGRADIENTE

O objectivo deste anexo é apresentar as técnicas de optimização subgradiente utilizadas neste trabalho para resolver problemas duais Lagrangeanos.

A primeira aplicação destes métodos no contexto da dualidade Lagrangeana deve-se a Held e Karp (1971) e revestiu-se de um enorme sucesso. Foi com esta ferramenta que o algoritmo para resolver o problema do caixeiro viajante concebido por estes autores se tornou bastante eficaz. Costumam referir-se como antecedentes teóricos destes métodos os trabalhos de investigadores soviéticos da década de 60 (Shor, Demyanov e Polyac, entre outros), e numa outra vertente aplicada à resolução de sistemas de desigualdades lineares os trabalhos de Motzkin e Schoenberg (1954) e Among (1954).

As demonstrações rigorosas da grande maioria dos resultados apresentados neste anexo podem encontrar-se em Shapiro (1979).

A maximização de uma função diferenciável h, de R^m em R, num subconjunto compacto de R^m,S, é com frequência feita pela determinação de uma sucessão $\{x_K\}$ de pontos de S. Esta sucessão é gerada obtendo-se x_{K+1} em função de x_K de acordo com a seguinte expressão:

$$x_{K+1} = x_K + t_K d_K \qquad (A.1)$$

onde d_K é uma direcção em R^m e t_K um passo.

O método do gradiente (steepest ascent), porventura o mais utilizado dos métodos de optimização diferenciável, define esta sequência tomando para direcção o gradiente de h no ponto x_K, $\nabla h(x_K)$, e para

passo a solução do problema

$$\underset{t_K}{\text{Max}}\ h\ (\ x_K + t_K\ \nabla h(x_K)\ ) \qquad (A.2)$$

$$\text{sa}\ x_K + t_K\ \nabla h(x_K) \in S. \qquad (A.3)$$

A sucessão $\{\ h(x_K)\ \}$ é monótona crescente e em certas condições é possivel garantir a convergência desta sucessão para h^*, o máximo de h.

Para resolver problemas da forma:

$$\underset{u\geq 0}{\text{Max}}\ w(u) \qquad (D)$$

$$\text{com } w(u) = \min_{x\in X}\ f(x)+ug(x)$$

onde f e g são funções contínuas de R^n em R e R^m, respectivamente, X é um conjunto finito e discreto de pontos de R^n e u um vector real de dimensão m (como é o caso dos duais Lagrangeanos em programação discreta), não é possível utilizar este método nem qualquer outro baseado na noção de gradiente pois a função w(u) não é diferenciável em toda a parte.

No entanto, a função dual, w(u), encerra características que tornam aconselhável a utilização de técnicas de optimização subgradiente. Estas técnicas naturalmente fazem apelo à noção de subgradiente podendo esta ser encarada como uma generalização do conceito de gradiente a funções côncavas. É possível mostrar que:

==========

TEOREMA

A função w(u) é côncava e finita em R.

==========

Pode então generalizar-se a noção de gradiente do seguinte modo:

DEFINIÇÃO

Sendo u, v e y vectores reais de dimensão m, y diz-se um subgradiente de w em v>0 se para todo o u>0 se tem

$$w(u)-w(v) \leqslant y(u-v).$$

DEFINIÇÃO

Ao conjunto de todos os subgradientes de w em v chama-se subdiferencial de w em v, Dw(v).

Podem ainda apresentar-se dois resultados relacionados com esta definição:

==========

TEOREMA

Para todo o u⩾0 Dw(u) é um convexo fechado de R^m.

==========

==========

TEOREMA

A função w é diferenciável no ponto u se e só se Dw(u) for um conjunto singular e nesse caso Dw(u)={∇w(u)}.

==========

É de acordo com este resultado que se refere a noção de subgradiente como uma generalização do conceito de gradiente.

O algoritmo de subgradiente respeita o esquema geral definido no início deste anexo gerando uma sequência de pontos $\{ u_K \}$ com a seguinte expressão:

$$u_{K+1} = P\ (u_K + t\ y_K) \qquad (A.4)$$

onde y_K é um subgradiente de w em u_K, P um projector de um ponto de R^m num ponto de R^m_+, e são dados u_o e uma sequência de escalares positivos. O projector usado é geralmente o de menor distância Euclideana, e o $u_0=0$.

É necessário discutir agora duas questões essenciais no sentido de precisar a descrição acima feita. A primeira prende-se com a escolha do subgradiente a utilizar para direcção de deslocamento a partir de u_K, estando esta questão fortemente relacionada com as diferenças fundamentais entre estes métodos e o método gradiente. A segunda relaciona-se com a escolha da sequência {tk} da qual dependem a maior parte dos resultados teóricos conhecidos, sendo também esta a questão onde se centra a maior parte da discussão sobre estes algoritmos.

No seguimento serão apresentados alguns resultados e definições necessários á justificação da escolha do subgradiente. Numa primeira fase será mostrado como é dificil obter o subgradiente que assegura o maior crescimento local da função, isto é, a selecção de um subgradiente a satisfazer os requisitos do gradiente. Depois apresentar-se-ão dois resultados que justificam finalmente a opção tomada.

DEFINIÇÃO

Sendo d um vector real de dimensão m, a derivada direccional de w segundo d em u é

$$w'(u;d) = \lim_{t \to 0} \frac{w(u+td)-w(u)}{t}$$

Pode ainda provar-se que

==========

TEOREMA

$$w'(u;d) = \min_{y \in Dw(u)} dy$$

==========

Então d será uma direcção de crescimento local de w se $w'(u;d)>0$ e a direcção de maior crescimento local da função resulta da solução de

$$\max_{||d||<1} w'(u;d) = \max_{||d||<1} \min_{y \in Dw(u)} yd = \min_{y \in Dw(u)} ||y||$$

ou seja, é dada pelo subgradiente de menor norma Euclideana, sendo então uma condição necessária e suficiente para que w(u) seja o máximo de w que $0 \in dw(u)$.

A determinação do subgradiente que assegura o maior crescimento local da função está assim dependente do conhecimento completo do subdiferencial, e este na maior parte dos casos fora de alcance em termos práticos. Os dois teoremas seguintes permitem ultrapassar este impasse.

TEOREMA

Seja X(u) o conjunto das soluções óptimas de

$$w(u) = \min_{x \in X} \{f(x)+ug(x)\}$$

isto é,

$$X(u) = \{\tilde{x} \in X: f(\tilde{x})+ug(\tilde{x})=w(u)\}$$

e seja

$$Y(u) = \{y \in R^m : y = g(\tilde{x}),\ \tilde{x} \in X(u)\}$$

então $Dw(u)=ConvY(u)$.

Deste teorema infere-se que a obtenção de um valor da função dual fornece automáticamente um subgradiente de w em u, isto é, se $w(u) = f(\tilde{x})+ug(\tilde{x})$, então $g(\tilde{x})$ é um subgradiente de w em u. É precisamente este subgradiente que é tomado para direcção de deslocamento. No entanto não existe qualquer garantia de que esta seja uma direcção de crescimento local da função mas esta opção é justificada pelo seguinte resultado:

TEOREMA

Seja u^* o ponto onde é atingido o máximo de w(u) e $d(u)=||u-u^*||$, $u \neq u^*$. Nestas condições qualquer subgradiente de w em u é uma direcção de descida para d(u).

Quer isto dizer que a sucessão de valores $\{u_K\}$ determinada pela expressão (A.4) é monótona em termos de aproximação ao maximizante de w. Sendo neste ponto que começam as diferenças fundamentais entre estes métodos e o método gradiente, pois neste último o que se garantia era a monotonia em termos de aproximação ao máximo.

Resolvida a questão da direcção, falta discutir qual a sucessão de reais positivos a utilizar para finalmente se concretizar o algoritmo subgradiente. A filosofia subjacente a este método pode sugerir um procedimento equivalente ao do método gradiente, isto é, determinar o passo em cada iteração de forma a que o novo ponto, u_{K+1}, seja o mais próximo possivel de u*. Ou seja, procurando-se a solução do problema:

$$\min_{t_K} || u^*-[u_K+t_K g_K(\tilde{x})] ||^2 \qquad (A.5)$$

que com facilidade se verifica ser

$$t_K^* = \frac{g_K(\tilde{x})(u^*-u_K)}{||g_K(\tilde{x})||^2} \qquad (A.6)$$

Mas este procedimento só é possivel mediante o conhecimento de u*, o que é um absurdo. No entanto, como

$$t_K^* = \frac{g_K(\tilde{x})(u^*-u_K)}{||gk(\tilde{x})||^2} > \frac{w^*-w(u_K)}{||g_K(\tilde{x})||^2} \qquad (A.7)$$

pode aceitar-se como uma estimativa de t *

$$t_K = \frac{\bar{w} - w(u_K)}{||g_K(\tilde{x})||^2} \qquad (A.8)$$

onde w é um majorante de w*.

Contudo a determinação do passo desta forma não permite demonstrar a convergência do algoritmo.

Polyac demonstrou que as seguintes condições :

$$t_K ||g_K(\tilde{x})|| \to 0 \text{ e } \sum_k t_K ||g_K(\tilde{x})|| = \infty \qquad (A.9)$$

são condições suficientes para se garantir a convergência do algoritmo para o óptimo w(u*). Nem todas as concretizações deste resultado conduzem a boas regras pois podem chegar-se a situações de

convergência muito lenta. É ainda Polyac que demonstra que tomando

$$t_K = p_K \frac{[w^*-w(u_K)]}{||g_K(\tilde{x})||^2} \qquad \text{com} \quad 0 < p_K < 2 \tag{A.10}$$

se consegue garantir a convergência do algoritmo para w^* e, sob certas condições esta convergência se faz geométricamente. No entanto, na maior parte dos casos, não é conhecido w^* e portanto este valor deve ser substituido por uma estimativa, perdendo-se assim a garantia de convergência.

Na literatura encontram-se inúmeras aplicações dos métodos subgradiente com resultados bastante razoáveis. As regras práticas de determinação do passo são as mais variadas, mas não verificam condições de convergência.

Held e Karp (1971) tomam $t_K=1$ para todo o k. Em Held, Wolfe e Crowder (1974), um artigo fundamental sobre estes métodos, propõe-se a seguinte expressão para gerar os valores de t_K:

$$t_K = p_K \frac{[\bar{w}-w(u_K)]}{||g_K(\tilde{x})||^2} \tag{A.11}$$

onde $\bar{w} \geqslant w^*$ e p_K toma o valor 2 nas 2n primeiras iterações (n dimensão do problema), sendo sucessivamente divididos por 2 tanto p_K como o número de iterações até que estas atinjam um valor pré-fixado $\bar{k}$; p_K é então dividido por 2 de $\bar{k}$ em $\bar{k}$ iterações até que o t_K resultante seja suficientemente pequeno. Os testes realizados apresentam resultados bastante satisfatórios. Algumas variantes desta regra têm sido também utilizadas com bons resultados.

No artigo de Bazaraa e Sherali (1981) apresenta-se uma regra original, contendo duas fases distintas sendo a segunda fase dedicada à obtenção de convergência. Contudo os testes realizados por estes autores não permitem apresentar esta alternativa como preferível à anterior parecendo não se justificar a sofisticação inerente a esta regra.

Não seria correcto concluir este anexo sem salientar a extrema simplicidade de que se revestem estes métodos.

ANEXO 2 - CONJUNTO DE PROBLEMAS TESTADOS

Neste anexo descreve-se o modo como foram obtidos os 73 problemas que serviram para efectuar os testes computacionais realizados com o objectivo de avaliar os métodos em estudo neste trabalho.

Na literatura encontrou-se apenas um problema apresentado por Lokin (1978). Este problema refere-se a um exemplo concreto de ordenação de tarefas e contém 24 cidades e 7 classes.

Os restantes 72 problemas utilizados foram gerados de modo a proporcionarem uma análise dos efeitos de diferentes dimensões e estruturas de dados no desempenho dos diversos métodos.

Foram assim gerados problemas com as seguintes dimensões: 80 cidades com 6 e 12 classes, 100 cidades com 8 e 15 classes, 120 cidades com 9 e 18 classes e 150 cidades com 11 e 22 classes. Este conjunto de dimensões foi sugerido por Jongens e Volgenant (1985), e foi escolhido por incluir também diferentes proporções número de cidades/ número de classes. Com cada uma das 8 dimensões acima referidas foram gerados 9 problemas.

Procurando cobrir-se as diversas aplicações foram gerados problemas com 3 tipos de dados com as seguintes características particulares:

DT0 - cidades de classes distintas podem ter a mesma localização;

DT1 - o estabelecimento do conjunto das cidades que pertencem à mesma classe é completamente indiferente às relações de proximidade entre cidades;

DT2 - a distância entre duas cidades de uma mesma classe é majorada por uma determinada constante.

Concretamente os problemas com dados do tipo DT0 foram gerados do seguinte modo:

1- geraram-se aleatoriamente as coordenadas de n/2 pontos distintos do plano. Estas coordenadas são inteiros entre 1 e 100 para os problemas com 80 e 100 cidades, e entre 1 e 250 nos problemas com 120 e 150 cidades.

2- Para determinar a cardinalidade das classes, isto é , um conjunto de valores $(n_1, n_2, \ldots, n_m)$, geraram-se aleatoriamente n inteiros entre 1 e m, e somaram-se as ocorrências de cada um destes valores. Quando este processo dava origem a uma ou mais classes vazias era repetido.

3- Finalmente preencheram-se as classes seleccionando-se aleatoriamente um conjunto de pontos entre os determinados em 1. Houve o cuidado de verificar se para uma mesma classe se teriam seleccionado duas cidades com a mesma localização o que nunca aconteceu.

Este tipo de dados foi concebido para gerar problemas com uma estrutura semelhante ao apresentado por Lokin.

Os dados do tipo DT1 foram gerados de modo a que a sua estrutura retratasse situações em que as classes são estabelecidas por motivos completamente alheios à proximidade geográfica das cidades que incluem. Os problemas com este tipo de dados foram obtidos do seguinte modo:

1- Determinou-se a cardinalidade de cada uma das m classes como no tipo de dados DT0 (conforme se explica no ponto 2 anterior).

2- Geram-se aleatoriamente as coordenadas (inteiros entre 1 e 1000) de n pontos do plano. Os primeiros n_1 pontos correspondem às localizações das cidades da primeira classe, os n_2 pontos seguintes às localizações das cidades da segunda classe, e assim por diante.

Os dados do tipo DT2 foram concebidos para representarem situações em que as classes são definidas de acordo com a proximidade das cidades que incluem. Os dados deste tipo foram gerados com a seguinte sequência de passos:

1- Determinou-se a cardinalidade de cada uma das classes como nos tipos de dados anteriores.

2- a determinação da localização das cidades de cada classe fez-se em 2 passos:
   a) geraram-se as coordenadas inteiras de um ponto de um quadrado de lado entre 1 e 1000.
   b) Para os problemas com 80 cidades desenhou-se um quadrado de lado entre 1 e 50 cujo ponto central é o que se determinou na alínea anterior. Neste quadrado geraram-se as coordenadas (inteiras) dos pontos que representam a localização das restantes cidades de cada classe. Para os problemas com 100 cidades e com mais de 100 cidades tomou-se respectivamente quadrados de lado entre 1 e 100 e 1 e 250.

Os números aleatórios foram gerados com base no gerador do VAX. Consideraram-se as distâncias euclideanas, isto é, a distância entre as cidades (x_i, y_j) e (x_j, y_j) foi calculada de acordo com a seguinte expressão:

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Para cada um dos tipos de dados foram gerados 3 problemas com cada uma das dimensões referidas no inicio do anexo. No conjunto de 73 problemas estão assim incluidos 25 problemas com dados do tipo DT0 e 24 problemas com cada um dos restantes tipos de dados.

ANEXO - 3 LISTAGEM DOS PROGRAMAS

Este anexo contém a listagem dos programas utilizados neste trabalho.

A utilização destes programas pressupõe a criação de um ficheiro, dados.dat com os seguintes elementos de input:

nº de cidades nº de classes
classe a que pertence cada uma das cidades
nº real superior a todas as distâncias entre cidades
elementos do bloco triângular superior da matriz de distâncias simétrica (excluindo os elementos da diagonal principal)

Depois da listagem de cada programa apresenta-se um exemplo de output correspondente ao seguinte ficheiro de input:

```
13 5
1 1 1 2 2 3 4 4 4 4 5 5
100000
2.82843E+02 5.00000E+02 7.00000E+02 1.62788E+03 1.60000E+03
8.94427E+02 1.11803E+03 1.39284E+03 1.06301E+03 7.28011E+02
6.32456E+02 5.00000E+02 3.60555E+02 5.38516E+02 1.40357E+03
1.41421E+03 6.32456E+02 8.54400E+02 1.14018E+03 7.81025E+02
5.00000E+02 4.00000E+02 2.23607E+02 2.00000E+02 1.14018E+03
1.10000E+03 5.00000E+02 7.07107E+02 9.43398E+02 7.61577E+02
2.82843E+02 6.70820E+02 4.47214E+02 9.48683E+02 9.00000E+02
4.12311E+02 5.83095E+02 7.81025E+02 7.07107E+02 2.00000E+02
7.81025E+02 5.65685E+02 3.00000E+02 8.06226E+02 6.32456E+02
3.60555E+02 8.94427E+02 9.05539E+02 1.43178E+03 1.30384E+03
8.94427E+02 7.81025E+02 5.83095E+02 1.06301E+03 9.21954E+02
1.52315E+03 1.36015E+03 2.23607E+02 5.09902E+02 3.00000E+02
2.23607E+02 6.32456E+02 5.00000E+02 3.00000E+02 2.82843E+02
4.24264E+02 8.06226E+02 7.07107E+02 5.38516E+02 6.70820E+02
1.10454E+03 1.00499E+03 5.09902E+02 6.08276E+02 5.83095E+02
6.40312E+02 4.47214E+02 2.23607E+02
```

Este ficheiro contém os dados do problema com 13 cidades e 5 classes que serviu para elaborar as ilustrações dos métodos heurísticos.

A.3.1 - LISTAGEM DOS PROGRAMAS DOS MÉTODOS HEURÍSTICOS

A.3.1.1 - PROGRAMA DO MÉTODO DO VIZINHO MAIS PRÓXIMO CLASSIFICADO

VARIÁVEIS DE INPUT

DADOS - ficheiro de input que contém o número de cidades, o número de classes, a classe a que pertence cada uma das cidades,um real superior a qualquer das distâncias e a matriz de distâncias;

N - número de cidades;

NUMCL - número de classes;

INF - real superior a qualquer das distâncias;

CL - vector que na posição i contém a classe a que pertence a cidade i;

C - vector que contém as distâncias entre cidades.

VARIÁVEIS DE OUTPUT

PERCTOT - vector que contém o percurso determinado;

PESOTOT - valor da solução.

LISTAGEM DO PROGRAMA:

```
PROGRAM MVPC(input,output,dados);

(* ESTE PROGRAMA DETERMINA SOLUÇÕES PARA O PROBLEMA DO CAIXEIRO
   VIAJANTE CLASSIFICADO PELO MÉTODO DO VIZINHO MAIS PRÓXIMO
   CLASSIFICADO. ESTE MÉTODO É APLICADO N VEZES E SELECCIONA-SE
   A MELHOR SOLUÇÃO *)

CONST n1=150; (*num de nodos*)
      m1=(sqr(n1)-n1)div 2;(*num de arestas (sqr(n)-n)/2) *)
      nncl1=25; (*max do num nodos de classe *)
      numcl1=30; (*num de classes *)
      numclq=(sqr(numcl1)-numcl1)div 2; (* (sqr(numcl)-numcl)/2 *)
```

```
TYPE vecin=array[1..n1] of integer;
     vecrqn=array[1..m1] of real;
     vecinncl=array[1..nncl1] of integer;
     veciqnumcl=array[1..numclq] of integer;
     vec0inumcl=array[0..numcl1] of integer;
     vec0rqnumcl=array[0..numclq] of real;
     vecrnumcl=array[1..numcl1] of real;

VAR s,temp,n,numcl,ninicial,nncl,
    hiato,nncli,i,j,l,s1,s2,total                    :integer;
    x1,x2,x3,x4,inf,menor,pesot,pesocl,pesotot       :real;
    cl,perct,perctot                                 :vecin;
    c                                                :vecrqn;
    caminho                                          :vecinncl;
    clciclo,clcomp                                   :vec0inumcl;
    dados                                            :text;


FUNCTION ivec (I,J: integer):integer;

(* esta função recebe os índices de uma matriz (triângular
   superior I<J)  quadrada de  ordem n e  transforma-os no
   índice de um vector *)

var t:integer;

begin
  if i=j then ivec:=0;
  if i>j then
           begin
             t:=i;
             i:=j;
             j:=t
           end;
  ivec:=((i-1)*n-((1+i)*i)div 2 )+j
end; (*ivec*)




PROCEDURE ledados;

(* este prcedimento lê os dados de um ficheiro *)

begin
  reset(dados);
  read(dados,n);
  read(dados,numcl);
  for i:=1 to n do read(dados,cl[i]);
  read(dados,inf);
  for i:=1 to trunc((sqr(n)-n)/2) do read(dados,c[i])
end; (*ledados*)
```

```
PROCEDURE vizprox(
              s1,nncl,hiato      :integer;
              var s2             :integer;
              var pesocl         :real;
              var caminho        :vecinncl);

(* Este procedimento determina um caminho Hamiltoniano pelo
   método do vizinho mais próximo começando na cidade s1     *)
(* devolve em s2 a cidade em que termina o caminho *)

var i,j,ind,vizprox          :integer;
    distmin                  :real;
    ciclo                    :vecinncl;

begin
  for i:=1 to nncl do ciclo[i]:=0;
  pesocl:=0;
  ind:=s1-hiato;
  for i:=1 to nncl-1 do
    begin
      distmin:=inf;
      ciclo[ind]:=ind;
      for j:=1 to nncl do
       if ciclo[j]=0 then
         if c[ivec(hiato+ind,hiato+j)]<distmin then
           begin
             distmin:=c[ivec(hiato+ind,hiato+j)];
             vizprox:=j
           end;
      pesocl:=pesocl+distmin;
      ciclo[ind]:=vizprox;
      ind:=vizprox;
    end;
  ind:=s1-hiato;
  for i:=1 to nncl do
    begin
      caminho[i]:=ind;
      ind:=ciclo[ind]
    end;
  for i:=1 to nncl do caminho[i]:=caminho[i]+hiato;
  s2:=caminho[nncl]

end; (*vizprox*)


(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)


begin
  x3:=clock;
  ledados;
  x1:=clock;
```

```
(* construir o vector clcomp que na posição i terá o número
   de cidades desde a classe 1 até à classe i *)

clcomp[0]:=0;
for i:=1 to numcl do clcomp[i]:=0;
for i:=1 to n do clcomp[cl[i]]:=clcomp[cl[i]]+1;
for i:=1 to numcl do clcomp[i]:=clcomp[i-1]+clcomp[i];
pesotot:=inf*n;
for s:=1 to n do
  begin
    for i:=1 to numcl do clciclo[i]:=0;
    pesot:=0;
    ninicial:=s;
    s1:=ninicial;
    total:=1;
    for l:=1 to numcl-1 do
      begin
        nncli:=clcomp[cl[s1]]-clcomp[cl[s1]-1];
        if nncli<>1 then
          begin
            vizprox(s1,nncli,clcomp[cl[s1]-1],s2,pesocl,caminho);
            pesot:=pesot+pesocl;
            for i:=1 to nncli-1 do
              begin
                perct[total]:=caminho[i];
                total:=total+1
              end
          end
                    else s2:=s1;
        clciclo[cl[s1]]:=1;
        menor:=inf;
        perct[total]:=s2;
        total:=total+1;
        for i:=1 to n do
        if clciclo[cl[i]]=0 then
          if c[ivec(i,s2)]<menor then
                                    begin
                                      menor:=c[ivec(i,s2)];
                                      temp:=i
                                    end;
        pesot:=pesot+menor;
        s1:=temp
      end;
    if total=n then
                  begin
                    perct[total]:=s1;
                    pesot:=pesot+c[ivec(s1,ninicial)];
                  end
                else
                  begin
                    nncli:=clcomp[cl[s1]]-clcomp[cl[s1]-1];
                    vizprox(s1,nncli,clcomp[cl[s1]-1],s2,pesocl,
                                                        caminho);
                    pesot:=pesot+pesocl;
                    for i:=1 to nncli do
                      begin
                        perct[total]:=caminho[i];
                        total:=total+1
                      end;
```

```
                    pesot:=pesot+c[ivec(s2,ninicial)];
                  end;
                if pesot<pesotot then
                  begin
                    pesotot:=pesot;
                    for i:=1 to n do perctot[i]:=perct[i]
                  end
    end;
(*output*)

  x2:=clock;
  writeln('          MÉTODO DO VIZINHO MAIS PRÓXIMO CLASSIFICADO');
  writeln;
  writeln;
  write('      PERCURSO OBTIDO:');
  j:=trunc(n/20)+1;
  i:=1;
  for l:=1 to j do
    begin
      repeat
        write(perctot[i]:3);
        i:=i+1;
      until i=n+1;
      writeln;
    end;
  write('      O CUSTO DESTE PERCURSO É :  ');
  writeln(pesotot);
  pesotot:=0;
  for i:=1 to n-1 do pesotot:=pesotot+
                              c[ivec(perctot[i],perctot[i+1])];
  pesotot:=pesotot+c[ivec(perctot[n],perctot[1])];
  writeln;
  writeln;
  writeln('      VERIFICAÇÃO DO CUSTO DO PERCURSO ',pesotot);
  writeln;
  writeln('      tempo de cpu sem io e ',(x2-x1)/1000, ' segundos');
  x4:=clock;
  writeln('      tempo de cpu com io e ',(x4-x3)/1000, ' segundos');
end.
```

EXEMPLO DE OUTPUT:

```
  MÉTODO DO VIZINHO MAIS PRÓXIMO CLASSIFICADO


PERCURSO OBTIDO: 12 13  2  1  3  4  5  6  9  8  7 11 10
O CUSTO DESTE PERCURSO É :   5.12723E+03


VERIFICAÇÃO DO CUSTO DO PERCURSO  5.12723E+03

tempo de cpu sem io e  7.00000E-02 segundos
tempo de cpu com io e  2.40000E-01 segundos
```

A.3.1.2 - PROGRAMA DO MÉTODO CLASSES-CIDADES

VARIÁVEIS DE INPUT

DADOS - ficheiro de input que contém o número de cidades, o número de classes, a classe a que pertence cada uma das cidades,um real superior a qualquer das distâncias e a matriz de distâncias;

N - número de cidades;

NUMCL - número de classes;

INF - real superior a qualquer das distâncias;

CL - vector que na posição i contém a classe a que pertence a cidade i;

C - vector que contém as distâncias entre cidades.

VARIÁVEIS DE OUTPUT

PERCT - vector que contém o percurso determinado;

PESOTOT - valor da solução.

LISTAGEM DO PROGRAMA:

```
PROGRAM MCLASCID(input,output,dados);

(* ESTE PROGRAMA DETERMINA UMA SOLUÇÃO PARA O PROBLEMA DO CAIXEIRO
   VIAJANTE CLASSIFICADO PELO MÉTODO CLASSES-CIDADES *)

CONST n1=150; (*num de nodos*)
      m1=(sqr(n1)-n1)div 2;(*num de arestas (sqr(n)-n)/2) *)
      nncl1=25; (*max do num nodos de uma classe*)
      numcl1=30; (*num de classes *)
      numclq=(sqr(numcl1)-numcl1)div 2; (* (sqr(numcl)-numcl)/2 *)
      numcl12=2*numcl1;(*2*num nodos de classe*)

TYPE vecin=array[1..n1] of integer;
     vecrqn=array[1..m1] of real;
     vecinncl=array[1..nncl1] of integer;
     veciqnumcl=array[1..numclq] of integer;
     vec0inumcl=array[0..numcl1] of integer;
     vec0rqnumcl=array[0..numclq] of real;
     vecrnumcl=array[1..numcl1] of real;
     veci2numcl=array[1..numcl12] of integer;
```

```
VAR n,numcl,nncl,hiato,nncli,i,j,k,l,s,s1,
    s2,t1,t2,total                                  :integer;
    x1,x2,x3,x4,inf,menor,pesot,
    pesocl,pesocl1,pesotot                          :real;
    cl,perct                                        :vecin;
    cidperc                                         :veci2numcl;
    c                                               :vecrqn;
    caminho,caminho1                                :vecinncl;
    a1,a2                                           :veciqnumcl;
    perc,perctot,clcomp                              :vec0inumcl;
    d                                               :vec0rqnumcl;
    dados                                           :text;

FUNCTION ivec (I,J: integer):integer;

(* esta função recebe os índices de uma matriz (triângular
   superior I<J) quadrada de ordem n e transforma-os no
   índice de um vector *)

VAR t:integer;

begin
  if i=j then ivec:=0;
  if i>j then
          begin
            t:=i;
            i:=j;
            j:=t
          end;
  ivec:=((i-1)*n-((1+i)*i)div 2 )+j
end; (*ivec*)


FUNCTION civec (I,J: integer):integer;

(* esta função recebe os índices de uma matriz (triângular
   superior I<J) quadrada de ordem numcl e transforma-os no
   índice de um vector *)

var t:integer;

begin
if i=j then civec:=0;
if i>j then
         begin
           t:=i;
           i:=j;
           j:=t
         end;
civec:=((i-1)*numcl-((1+i)*i)div 2 )+j
end; (*civec*)
```

```
PROCEDURE ledados;

(* este procedimento lê os dados de um ficheiro *)

begin
  reset(dados);
  read(dados,n);
  read(dados ,numcl);
  for i:=1 to n do read(dados,cl[i]);
  read(dados ,inf);
  for i:=1 to trunc((sqr(n)-n)/2) do read(dados,c[i])
end; (*ledados*)



PROCEDURE mia(
        numcl,s  :integer;
        inf      :real;
        d        :vec0rqnumcl;
        var perc :vec0inumcl;
        var pesot:real);


(* Este procedimento determina numcl circuitos  Hamiltonianos
   pelo método da inserção mais afastada e devolve o circuito
   de menor peso e o respectivo peso *)


var  nter1,nter2,mafast,ind,proxind,i,j   :integer;
     custins,maxdist,novocust             :real;
     ciclo                                :vec0inumcl;
     dist                                 :vecrnumcl;


begin
  for i:=1 to numcl do ciclo[i]:=0;
  ciclo[s]:=s;
  for i:=1 to numcl do dist[i]:=d[civec(i,s)];
  pesot:=inf;
  for i:=1 to numcl-1 do
    begin
      maxdist:=-inf;
      for j:=1 to numcl do
        if ciclo[j]=0 then
          if dist[j]>maxdist then
                                begin
                                  maxdist:=dist[j];
                                  mafast:=j
                                end;
      custins:=inf;
      ind:=s;
      nter2:=s;
```

```
      for j:=1 to numcl do
        begin
          proxind:=ciclo[ind];
          novocust:=d[civec(ind,mafast)]+d[civec(mafast,proxind)]
                                        -d[civec(ind,proxind)];
          if novocust<custins then
                                  begin
                                    custins:=novocust;
                                    nter1:=ind;
                                    nter2:=proxind
                                  end;
          ind:=proxind;
        end;
      ciclo[mafast]:=nter2;
      ciclo[nter1]:=mafast;
      pesot:=pesot+custins;
      for j:=1 to numcl do
        if ciclo[j]=0 then
          if d[civec(mafast,j)]<dist[j] then
                                         dist[j]:=d[civec(mafast,j)]
    end;
  ind:=s;
  for i:=1 to numcl do
    begin
      perc[i]:=ind;
      ind:=ciclo[ind]
    end;


end; (*mia*)


PROCEDURE vizprox(
              t1,t2,nncl,hiato :integer;
              var pesocl       :real;
              var caminho      :vecinncl);

(* Este procedimento determina caminhos Hamiltonianos pelo método
   do vizinho mais próximo com início em t1 e fim em t2 *)

var i,j,ind,vizprox  :integer;
    distmin          :real;
    ciclo            :vecinncl;

begin
  for i:=1 to nncl do ciclo[i]:=0;
  ciclo[t1-hiato]:=t1-hiato;
  ciclo[t2-hiato]:=t2-hiato;
  pesocl:=0;
  ind:=t1-hiato;
  for i:=1 to nncl-2 do
    begin
      ciclo[ind]:=ind;
      distmin:=inf;
```

```
      for j:=1 to nncl do
        if ciclo[j]=0 then
          if c[ivec(hiato+ind,hiato+j)]<distmin then
               begin
                 distmin:=c[ivec(hiato+ind,hiato+j)];
                 vizprox:=j
               end;
      pesocl:=pesocl+distmin;
      ciclo[ind]:=vizprox;
      ind:=vizprox
    end;
  ind:=t1-hiato;
  for i:=1 to nncl-1 do
    begin
      caminho[i]:=ind;
      ind:=ciclo[ind]
    end;
  pesocl:=pesocl+c[ivec(t2,caminho[nncl-1]+hiato)];
  for i:=1 to nncl-1 do caminho[i]:=caminho[i]+hiato;
  caminho[nncl]:=t2

end; (*vizprox*)


PROCEDURE rrgrau(
             i              :integer;
             c              :vecrqn;
             var pesotot    :real;
             var cidperc    :veci2numcl);

(* Este procedimento corrige o grau da cidade terminal
   da classe i *)

var temp,j,t1,t2  :integer;
    melhor        :real;
    a             :boolean;


begin
  if 2*i+1>2*numcl then t1:=1 else t1:=2*i+1;
  if 2*i-2<1 then t2:=2*numcl else t2:=2*i-2;
  melhor:=inf;
  for j:=clcomp[cl[cidperc[i*2]]-1]+1 to clcomp[cl[cidperc[i*2]]] do
    if j<>cidperc[i*2] then
      begin
        if c[ivec(j,cidperc[t1])]<melhor then
          begin
            melhor:=c[ivec(j,cidperc[t1])];
            temp:=j;
            a:=true
          end;
        if c[ivec(j,cidperc[t2])]<melhor then
          begin
            melhor:=c[ivec(j,cidperc[t2])];
            temp:=j;
            a:=false
          end
      end;
```

```
  if a then
        begin
          pesotot:=pesotot-c[ivec(cidperc[i*2],cidperc[t1])]
                           +c[ivec(cidperc[t1],temp)];
          cidperc[i*2]:=temp
        end
      else
        begin
          pesotot:=pesotot-c[ivec(cidperc[i*2-1],cidperc[t2])]
                           +c[ivec(cidperc[t2],temp)];
          cidperc[i*2-1]:=temp
        end
end; (*rrgrau*)


(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)

begin
  x3:=clock;
  ledados;
  x1:=clock;

  (* construir o vector clcomp, que na posição i terá o número
     de cidades desde a classe 1 até à classe i *)

  clcomp[0]:=0;
  for i:=0 to numcl do clcomp[i]:=0;
  for i:=1 to n do clcomp[cl[i]]:=clcomp[cl[i]]+1;
  for i:=1 to numcl do clcomp[i]:=clcomp[i-1]+clcomp[i];

  (*contruir a matriz de distâncias do grafo das classes*)

  d[0]:=inf;
  for l:=1 to numcl do
  for k:=l+1 to numcl do
    begin
      menor:=inf;
      for i:=clcomp[l-1]+1 to clcomp[l] do
        for j:=clcomp[k-1]+1 to clcomp[k] do
          if c[ivec(i,j)]< menor then
            begin
              menor:=c[ivec(i,j)];
              d[civec(l,k)]:=c[ivec(i,j)];
              a1[civec(l,k)]:=i;
              a2[civec(l,k)]:=j
            end;
    end;

(* determinar um circuito hamiltoniano no grafo das classes *)

  pesotot:=inf*numcl;
  pesot:=0;
  for s:=1 to numcl do
```

```
    begin
      mia(numcl,s,inf,d,perc,pesot);
      if pesot<pesotot then
        begin
          pesotot:=pesot;
          for i:=1 to numcl do perctot[i]:=perc[i];
        end
    end;
  j:=2;
  for i:=1 to numcl-1 do
  if perctot[i]<perctot[i+1] then
    begin
      cidperc[j]:=a1[civec(perctot[i],perctot[i+1])];
      cidperc[j+1]:=a2[civec(perctot[i],perctot[i+1])];
      j:=j+2
    end
                              else
    begin
      cidperc[j]:=a2[civec(perctot[i],perctot[i+1])];
      cidperc[j+1]:=a1[civec(perctot[i],perctot[i+1])];
      j:=j+2
    end;
  if perctot[1]<perctot[numcl] then
    begin
      cidperc[j]:=a2[civec(perctot[1],perctot[numcl])];
      cidperc[1]:=a1[civec(perctot[1],perctot[numcl])]
    end
                            else
    begin
      cidperc[j]:=a1[civec(perctot[1],perctot[numcl])];
      cidperc[1]:=a2[civec(perctot[1],perctot[numcl])]
    end;

(* alterar o grau das cidades de classes não
                                    singulares com grau 2 *)

  for i:=1 to numcl do
    if clcomp[perctot[i]]-clcomp[perctot[i]-1]<>1 then
      if cidperc[2*i-1]=cidperc[2*i] then
                                rrgrau(i,c,pesotot,cidperc);

(* determinar caminhos Hamiltonianos a unir as
                                    cidades de cada classe *)

  total:=1;
  for i:=1 to numcl do
    begin
      nncli:=clcomp[perctot[i]]-clcomp[perctot[i]-1];
      if nncli=1 then
        begin
          perct[total]:=cidperc[2*i-1];
          total:=total+1
        end
```

```
                else
        if nncli>2 then
          begin
            s1:=cidperc[2*i-1];
            s2:=cidperc[2*i];
            vizprox(s1,s2,nncli,clcomp[cl[s1]-1],pesocl,caminho);
            vizprox(s2,s1,nncli,clcomp[cl[s1]-1],pesocl1,caminho1);
            if pesocl<pesocl1 then
              begin
                l:=1;
                for j:=total to total+nncli-1 do
                  begin
                    perct[J]:=caminho[l];
                    l:=l+1
                  end;
                pesotot:=pesotot+pesocl;
              end
                              else
              begin
                l:=nncli;
                for j:=total to nncli+total-1 do
                  begin
                    perct[J]:=caminho1[l];
                    l:=l-1
                  end;
                pesotot:=pesotot+pesocl1;
              end;
            total:=total+nncli;
          end;
        if nncli=2 then
          begin
            pesotot:=pesotot+c[ivec(cidperc[2*i-1],cidperc[2*i])];
            perct[total]:=cidperc[2*i-1];
            perct[total+1]:=cidperc[2*i];
            total:=total+2
          end;
    end;

(*output*)

  x2:=clock;
  writeln('                    MÉTODO CLASSES CIDADES');
  writeln;
  writeln;
  write('      PERCURSO OBTIDO:');
  j:=trunc(n/20)+1;
  i:=1;
  for l:=1 to j do
    begin
      repeat
        write(perct[i]:3);
        i:=i+1;
      until i=n+1;
      writeln;
    end;
  write('      O CUSTO DESTE PERCURSO É :  ');
  writeln(pesotot);
```

```
  pesotot:=0;
  for i:=1 to n-1 do pesotot:=pesotot+c[ivec(perct[i],perct[i+1])];
  pesotot:=pesotot+c[ivec(perct[n],perct[1])];
  writeln;
  writeln;
  writeln('      VERIFICAÇÃO DO CUSTO DO PERCURSO ',pesotot);
  writeln;
  writeln('      tempo de cpu sem io e ',(x2-x1)/1000, ' segundos');
  x4:=clock;
  writeln('      tempo de cpu com io e ',(x4-x3)/1000, ' segundos');
end.
```

EXEMPLO DE OUTPUT:

```
        MÉTODO CLASSES CIDADES

PERCURSO OBTIDO:  3  1  2 12 13 11  7  8 10  9  6  5  4
O CUSTO DESTE PERCURSO É :   5.15402E+03

VERIFICAÇÃO DO CUSTO DO PERCURSO  5.15402E+03

tempo de cpu sem io e  1.90000E-01 segundos
tempo de cpu com io e  3.60000E-01 segundos
```

A.3.1.3 - PROGRAMA DO MÉTODO CIDADES-CLASSES

VARIÁVEIS DE INPUT

DADOS - ficheiro de input que contém o número de cidades, o número de classes, a classe a que pertence cada uma das cidades,um real superior a qualquer das distâncias e a matriz de distâncias;

N - número de cidades;

NUMCL - número de classes;

INF - real superior a qualquer das distâncias;

CL - vector que na posição i contém a classe a que pertence a cidade i;

C - vector que contém as distâncias entre cidades.

VARIÁVEIS DE OUTPUT

PERCFINAL- vector que contém o percurso determinado;

PESOTOT - valor da solução.

LISTAGEM DO PROGRAMA:

```
PROGRAM MCIDCLAS(input,output,dados);

(* ESTE PROGRAMA DETERMINA UMA SOLUÇÃO PARA O PROBLEMA DO CAIXEIRO
   VIAJANTE CLASSIFICADO PELO MÉTODO CIDADES CLASSES *)

CONST n1=150; (*num de nodos*)
      m1=(sqr(n1)-n1)div 2;(*num de arestas (sqr(n)-n)/2) *)
      nncl1=25; (*max do num nodos de classe *)
      numcl1=30; (*num de classes *)
      numclq=(sqr(numcl1)-numcl1)div 2; (* (sqr(numcl)-numcl)/2 *)
      numcl2=2*numcl1;(*2*numcl*)

TYPE vecin=array[1..n1] of integer;
     vec0rqn=array[0..m1] of real;
     vecinncl=array[1..nncl1] of integer;
     veciqnumcl=array[1..numclq] of integer;
     vec0inumcl=array[0..numcl1] of integer;
     vec0rqnumcl=array[0..numclq] of real;
     vecrnumcl=array[1..numcl1] of real;
     veci2numcl=array[1..numcl2] of integer;

VAR n,numcl,nncl,hiato,nncli,i,j,k,
    l,s,s1,s2,t1,t2,t,total                    :integer;
    x1,x2,x3,x4,pesonl,itpesonl,inf,
    menor,pesot,pesocl,pesotot                 :real;
    perct,cl,percfinal                         :vecin;
    c                                          :vec0rqn;
    caminho                                    :vecinncl;
    a1,a2                                      :veciqnumcl;
    percot,clcomp                              :vec0inumcl;
    d                                          :vec0rqnumcl;
    dados                                      :text;
    cidperc,perc                               :veci2numcl;

FUNCTION ivec (I,J: integer):integer;

(* esta função recebe os índices de uma matriz (triângular
   superior I<J) quadrada de ordem n  e  transforma-os  no
   índice de um vector *)

VAR t:integer;

begin
  if i=j then ivec:=0
         else
           if i>j then
                    begin
                      t:=i;
                      i:=j;
                      j:=t
                    end;
  ivec:=((i-1)*n-((1+i)*i)div 2 )+j
end; (*ivec*)
```

```
FUNCTION civec (I,J: integer):integer;

(* esta função recebe os índices de uma  matriz (triângular
   superior I<J) quadrada de ordem numcl e transforma-os no
   índice de um vector *)

VAR t:integer;

begin
  if i=j then civec:=0
         else
  if i>j then
           begin
             t:=i;
             i:=j;
             j:=t
           end;
  civec:=((i-1)*numcl-((1+i)*i)div 2 )+j
end; (*civec*)

PROCEDURE ledados;

(* este procedimento lê os dados de um ficheiro *)

begin
  reset(dados);
  read(dados,n);

  read(dados,numcl);
  for i:=1 to n do read(dados,cl[i]);
  read(dados,inf);
  for i:=1 to (sqr(n)-n)div 2 do read(dados,c[i]);
  c[0]:=inf
end; (*ledados*)


PROCEDURE mia(
     numcl,s   :integer;
     inf       :real;
     var d     :vec0rqnumcl;
     var perc  :veci2numcl;
     var pesot :real);

(* Este procedimento determina um circuito Hamiltoniano pelo
   método da inserção mais afastada começando na classe s*)

var  nter1,nter2,mafast,ind,proxind,i,j   :integer;
     custins,maxdist,novocust             :real;
     ciclo                                :vec0inumcl;
     dist                                 :vecrnumcl;

begin
  for i:=1 to numcl do ciclo[i]:=0;
  ciclo[s]:=s;
  for i:=1 to numcl do dist[i]:=d[civec(i,s)];
  pesot:=inf;
  for i:=1 to numcl-1 do
```

```
    begin
      maxdist:=-inf;
      for j:=1 to numcl do
      if ciclo[j]=0 then
        if dist[j]>maxdist then
                              begin
                                maxdist:=dist[j];
                                mafast:=j
                              end;
      custins:=inf;
      ind:=s;
      nter2:=s;
      for j:=1 to numcl do
        begin
          proxind:=ciclo[ind];
          novocust:=d[civec(ind,mafast)]+d[civec(mafast,proxind)]
                                        -d[civec(ind,proxind)];
          if novocust<custins then
                                 begin
                                   custins:=novocust;
                                   nter1:=ind;
                                   nter2:=proxind
                                 end;
          ind:=proxind;
        end;
      ciclo[mafast]:=nter2;
      ciclo[nter1]:=mafast;
      pesot:=pesot+custins;
      for j:=1 to numcl do
      if ciclo[j]=0 then
        if d[civec(mafast,j)]<dist[j] then
                                        dist[j]:=d[civec(mafast,j)]
    end;
  ind:=s;
  for i:=1 to numcl do
    begin
      perc[i]:=ind;
      ind:=ciclo[ind]
    end;

end;(*mia*)



PROCEDURE vizprox(
              nncl,hiato    :integer;
              var pesocl    :real;
              var caminho   :vecinncl);

(* Este procedimento determina nncl caminhos hamiltonianos pelo
   método do vizinho mais próximo e selecciona o de menor peso *)

var i,j,ind,vizprox        :integer;
    distmin,itpesocl       :real;
    ciclo                  :vecinncl;
```

```
begin
  pesocl:=inf*nncl;
  for s:=1 to nncl do
    begin
      for i:=1 to nncl do ciclo[i]:=0;
      itpesocl:=0;
      ind:=s;
      for i:=1 to nncl-1 do
        begin
          distmin:=inf;
          ciclo[ind]:=ind;
          for j:=1 to nncl do
            if ciclo[j]=0 then
              if c[ivec(hiato+ind,hiato+j)]<distmin then
                begin
                  distmin:=c[ivec(hiato+ind,hiato+j)];
                  vizprox:=j
                end;
          itpesocl:=itpesocl+distmin;
          ciclo[ind]:=vizprox;
          ind:=vizprox
        end;
        if itpesocl<pesocl then
                          begin
                            ind:=s;
                            for i:=1 to nncl do
                              begin
                                caminho[i]:=ind;
                                ind:=ciclo[ind]
                              end;
                            pesocl:=itpesocl
                          end;
    end;
  for i:=1 to nncl do caminho[i]:=caminho[i]+hiato;

end; (*vizprox*)


PROCEDURE rrgrau(var pesotot   :real);

(* Este procedimento corrige o grau as cidades terminais
                                                de uma classe *)

var t1,t2,t3:integer;

begin
  if 2*i-1=1 then t1:=2*numcl else t1:=2*i-2;
  if 2*i=2*numcl then t2:=1 else t2:=2*i+1;
  t3:=2*percot[i];
  if perc[2*i]=cidperc[t3] then
   if c[ivec(perc[t1],cidperc[t3-1])]-c[ivec(perc[t1],perc[2*i])]<
      c[ivec(perc[t2],cidperc[t3-1])]-c[ivec(perc[t2],perc[2*i])]
     then
       begin
         pesotot:=pesotot+c[ivec(perc[t1],cidperc[t3-1])]-
                          c[ivec(perc[t1],perc[2*i])];
         perc[2*i-1]:=cidperc[t3-1];
       end
```

```
      else
        begin
          pesotot:=pesotot+c[ivec(perc[t2],cidperc[t3-1])]-
                          c[ivec(perc[t2],perc[2*i])];
          perc[2*i]:=cidperc[t3-1];
        end
                          else

    if c[ivec(perc[t1],cidperc[t3])]-c[ivec(perc[t1],perc[2*i])]<
       c[ivec(perc[t2],cidperc[t3])]-c[ivec(perc[t2],perc[2*i])]
      then
        begin
          pesotot:=pesotot+c[ivec(perc[t1],cidperc[t3])]-
                          c[ivec(perc[t1],perc[2*i])];
          perc[2*i-1]:=cidperc[t3];
        end
      else
        begin
          pesotot:=pesotot+c[ivec(perc[t2],cidperc[t3])]-
                          c[ivec(perc[t2],perc[2*i])];
          perc[2*i]:=cidperc[t3];
        end;

end; (*rrgrau*)



(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)

begin
  x3:=clock;
  ledados;
  x1:=clock;
  (*construir o clcomp que na posição i terá o número
    de cidades desde a classe 1 até à classe i *)

  clcomp[0]:=0;
  for i:=1 to numcl do clcomp[i]:=0;
  for i:=1 to n do clcomp[cl[i]]:=clcomp[cl[i]]+1;
  for i:=1 to numcl do clcomp[i]:=clcomp[i-1]+clcomp[i];

  (* Determinar caminhos Hamiltonianos a unir as cidades
                                                 das classes *)

  pesotot:=0;
  total:=1;
  for i:=1 to numcl do
    begin
      nncli:=clcomp[i]-clcomp[i-1];
      case nncli of
        1: begin
             perct[total]:=clcomp[i];
             total:=total+1;
             cidperc[2*i-1]:=clcomp[i];
             cidperc[2*i]:=clcomp[i]
           end;
```

```
        2: begin
             perct[total]:=clcomp[i-1]+1;
             perct[total+1]:=clcomp[i];
             pesotot:=pesotot+c[ivec(clcomp[i],clcomp[i-1]+1)];
             total:=total+2;
             cidperc[2*i-1]:=clcomp[i-1]+1;
             cidperc[2*i]:=clcomp[i]
           end;
otherwise  begin
             vizprox(nncli,clcomp[i-1],pesocl,caminho);
             pesotot:=pesotot+pesocl;
             l:=1;
             for j:=total to nncli+total-1 do
               begin
                 perct[total]:=caminho[l];
                 total:=total+1;
                 l:=l+1;
               end;
             cidperc[2*i-1]:=caminho[1];
             cidperc[2*i]:=caminho[nncli];
           end;
      end;(*case*)

    end;
  (* construir a matriz de distâncias do grafo das classes *)
  d[0]:=inf;
  for i:=1 to numcl do
    begin
      for j:=i+1 to numcl do
        begin
          a1[civec(i,j)]:=cidperc[2*i-1];
          a2[civec(i,j)]:=cidperc[2*j-1];
          menor:=c[ivec(cidperc[i*2-1],cidperc[2*j-1])];
          if c[ivec(cidperc[i*2-1],cidperc[2*j])]<menor then
            begin
              menor:=c[ivec(cidperc[i*2-1],cidperc[2*j])];
              a2[civec(i,j)]:=cidperc[2*j]
            end;
          if c[ivec(cidperc[i*2],cidperc[2*j-1])]<menor then
            begin
              a1[civec(i,j)]:=cidperc[2*i];
              a2[civec(i,j)]:=cidperc[2*j-1];
              menor:=c[ivec(cidperc[i*2],cidperc[2*j-1])];
            end;
          if c[ivec(cidperc[i*2],cidperc[2*j])]<menor then
            begin
              a1[civec(i,j)]:=cidperc[2*i];
              a2[civec(i,j)]:=cidperc[2*j];
              menor:=c[ivec(cidperc[i*2],cidperc[2*j])];
            end;
          d[civec(i,j)]:=menor;
          if a1[civec(i,j)]>a2[civec(i,j)] then
            begin
              t:=a1[civec(i,j)];
              a1[civec(i,j)]:=a2[civec(i,j)] ;
              a2[civec(i,j)]:=t
            end;
        end
    end;
```

```
(* Determinar um circuito Hamiltoniano no grafo das classes *)
pesonl:=inf*numcl;
for s:=1 to numcl do
  begin
    mia(numcl,s,inf,d,perc,pesot);
    if pesot<pesonl then
                      begin
                        pesonl:=pesot;
                        for i:=1 to numcl do percot[i]:=perc[i]
                      end;
  end;
pesotot:=pesotot+pesonl;

j:=2;
for i:=1 to numcl-1 do
if percot[i]<percot[i+1] then
      begin
        perc[j]:=a1[civec(percot[i],percot[i+1])];
        perc[j+1]:=a2[civec(percot[i],percot[i+1])];
        j:=j+2
      end
                       else
      begin
        perc[j]:=a2[civec(percot[i],percot[i+1])];
        perc[j+1]:=a1[civec(percot[i],percot[i+1])];
        j:=j+2
      end;
if percot[1]<percot[numcl] then
      begin
        perc[j]:=a2[civec(percot[1],percot[numcl])];
        perc[1]:=a1[civec(percot[1],percot[numcl])]
      end
                       else
      begin
        perc[j]:=a1[civec(percot[1],percot[numcl])];
        perc[1]:=a2[civec(percot[1],percot[numcl])]
      end;

(* corrigir o grau das cidades terminais de cada classe *)
for i:=1 to numcl do
if clcomp[percot[i]]-clcomp[percot[i]-1]<>1 then
  if perc[2*i-1]=perc[2*i] then rrgrau(pesotot);
l:=1;

for i:=1 to numcl do
  begin
    nncli:=clcomp[percot[i]]-clcomp[percot[i]-1];
    case nncli of
          1: begin
               percfinal[l]:=perc[2*i];
               l:=l+1
             end;
          2: begin
               percfinal[l]:=perc[2*i-1];
               percfinal[l+1]:=perc[2*i];
               l:=l+2
             end;
```

```
    otherwise begin
                if perc[2*i-1]=cidperc[percot[i]*2-1] then
                  begin
                    for j:=1 to nncli do
                      begin
                        percfinal[l]:=perct[clcomp[percot[i]-1]+j];
                        l:=l+1
                      end;
                  end
                                                        else
                  begin
                    for j:=nncli downto 1 do
                      begin
                        percfinal[l]:=perct[clcomp[percot[i]-1]+j];
                        l:=l+1
                      end;
                  end
            end
      end;(*case*)
   end;
(*output*)

  x2:=clock;
  writeln('                MÉTODO CIDADES CLASSES');
  writeln;
  writeln;
  write('      PERCURSO OBTIDO:');
  j:=trunc(n/20)+1;
  i:=1;
  for l:=1 to j do
    begin
      repeat
        write(percfinal[i]:3);
        i:=i+1;
      until i=n+1;
      writeln;
    end;
  write('      O CUSTO DESTE PERCURSO É :  ');
  writeln(pesotot);
  pesotot:=0;
  for i:=1 to n-1 do pesotot:=pesotot+
                          c[ivec(percfinal[i],percfinal[i+1])];
  pesotot:=pesotot+c[ivec(percfinal[n],percfinal[1])];
  writeln;
  writeln;
  writeln('      VERIFICAÇÃO DO CUSTO DO PERCURSO ',pesotot);
  writeln;
  writeln('      tempo de cpu sem io e ',(x2-x1)/1000, ' segundos');
  x4:=clock;
  writeln('      tempo de cpu com io e ',(x4-x3)/1000, ' segundos');
end.
```

EXEMPLO DE OUTPUT:

```
        MÉTODO CIDADES CLASSES

PERCURSO OBTIDO:  3  2  1 13 12 10 11  7  8  9  6  5  4
O CUSTO DESTE PERCURSO É :   5.26418E+03

VERIFICAÇÃO DO CUSTO DO PERCURSO  5.26418E+03

tempo de cpu sem io e  4.00000E-02 segundos
tempo de cpu com io e  2.40000E-01 segundos
```

A.3.1.4 - PROGRAMA DO MÉTODO DE JONGENS E VOLGENANT

VARIÁVEIS DE INPUT

DADOS - ficheiro de input que contém o número de cidades, o número de classes, a classe a que pertence cada uma das cidades,um real superior a qualquer das distâncias e a matriz de distâncias;

N - número de cidades;

NUMCL - número de classes;

INF - real superior a qualquer das distâncias;

CL - vector que na posição i contém a classe a que pertence a cidade i;

C - vector que contém as distâncias entre cidades.

VARIÁVEIS DE OUTPUT

A - lista que contém o percurso determinado;

PESOTM - valor da solução.

LISTAGEM DO PROGRAMA:

```
PROGRAM MJV(input,output,dados);

(* ESTE PROGRAMA DETERMINA UMA SOLUÇÃO PARA O PROBLEMA DO CAIXEIRO
   VIAJANTE CLASSIFICADO PELO MÉTODO DE JONGENS E VOLGENANT *)

CONST n1=150;(*num de nodos*)
m1=(sqr(n1)-n1)div 2;(*num de arestas (sqr(n)-n)/2 *)
numcl1=30;(*num de classes *)


TYPE ponteiro=^lista;
     lista=record
              ant :ponteiro;
              x   :integer;
              prox:ponteiro;
           end;
     vecni=array[1..n1]of integer;
     vec0mr=array[0..m1]of real;
     vecnr=array[1..n1]of real;
     vecnumcli=array[1..numcl1]of integer;

VAR n,numcl,t,i,inicio,cest,gm,j,
    a1,b1,b2,a2,t1,a3,b3,b4,a4,t2                          :integer;
    l,temp,p,f,a,pt2,pt1,pb4,pb2,pb3,pb1,pa1,pa2,pa3,pa4:ponteiro;
    teste                                                  :boolean;
    x1,x2,x3,inf,pesotm,pesovp,custo                       :real;
    c                                                      :vec0mr;
    percvp,cl,perc                                         :vecni;
    g                                                      :vecnumcli;
    dados                                                  :text;


FUNCTION ivec (I,J: integer):integer;

(* esta funcao recebe os índices de uma matriz (triângular
   superior I<J)  quadrada  de  ordem n e transforma-os no
   índice de um vector *)

var t:integer;

begin
  if i=j then ivec:=0;
  if i>j then
           begin
             t:=i;
             i:=j;
             j:=t
           end;
  ivec:=((i-1)*n-((1+i)*i)div 2 )+j
end; (*ivec*)
```

```
PROCEDURE ledados;

(* este procedimento lê os dados de um ficheiro *)

begin
  reset(dados);
  read(dados,n);
  read(dados,numcl);
  for i:=1 to n do read(dados,cl[i]);
  read(dados ,inf);
  for i:=1 to trunc((sqr(n)-n)/2) do read(dados,c[i]);
  c[0]:=inf;
end;


PROCEDURE clista(perc:vecni);

(* este procedimento cria uma lista circular duplamente ligada *)

begin
  new(p);
  p^.ant:=nil;
  p^.prox:=nil;
  p^.x:=perc[1];
  new(f);
  f^.prox:=p;
  f^.ant:=nil;
  f^.x:=perc[n];
  p^.ant:=f;
  for i:=2 to n-1 do
    begin
      new(a);
      a^.x:=perc[i];
      a^.ant:=p;
      p^.prox:=a;
      a^.prox:=nil;
      p:=p^.prox;
    end;
  f^.ant:=a;
  a^.prox:=f;

end;(*clista*)


PROCEDURE seleccl;

(* este procedimento selecciona a classe a estudar *)

begin
  for i:=1 to numcl do g[i]:=0;
  for i:=1 to n do
    begin
      if cl[a^.x]<>cl[a^.prox^.x] then
        begin
          g[cl[a^.x]]:=g[cl[a^.x]]+1;
          g[cl[a^.prox^.x]]:=g[cl[a^.prox^.x]]+1;
        end;
      a:=a^.prox;
    end;
```

```
  i:=1;
  teste:=false;
  gm:=1000;
    repeat
      if g[i]=4 then
                  begin
                    teste:=true;
                    cest:=i;
                  end
                else
                  if (g[i]<gm) and (g[i]>2) then
                    begin
                      gm:=g[i];
                      cest:=i
                    end;
      i:=i+1;
    until (i=numcl+1) or teste ;
  if not(teste ) and (gm=1000) then gm:=2;

end; (*seleccl*)


PROCEDURE selecabt;

(* este procedimento selecciona as cidades ai, bi e ti *)

begin
  a1:=0;a2:=0;a3:=0;a4:=0;
    repeat
      if (cl[a^.x]=cest) and (cl[a^.prox^.x]<>cest) then
        begin
          a1:=a^.x;
          b1:=a^.prox^.x;
          pa1:=a
        end;
      a:=a^.prox;
    until a1<>0;
    repeat
      if (cl[a^.x]<>cest) and (cl[a^.prox^.x]=cest) then
        begin
          pa2:=a^.prox;
          a2:=a^.prox^.x;
          b2:=a^.x;
          t1:=pa2^.prox^.x;
        end;
      a:=a^.prox;
    until a2<>0;
    repeat
      if (cl[a^.x]=cest) and (cl[a^.prox^.x]<>cest) then
        begin
          a3:=a^.x;
          b3:=a^.prox^.x;
          pa3:=a;
        end;
      a:=a^.prox;
    until a3<>0;
```

```
    repeat
      if (cl[a^.x]<>cest) and (cl[a^.prox^.x]=cest) then
        begin
          pa4:=a^.prox;
          a4:=a^.prox^.x;
          b4:=a^.x;
          t2:=pa4^.prox^.x;
        end;
      a:=a^.prox;
    until a4<>0;

end;(*selectabt*)


PROCEDURE selecalt;

(* este procedimento selecciona a alternativa a executar *)

var teste1,teste2,teste3,teste4:boolean;

begin
  t:=0;
  teste1:=false;teste2:=false;teste3:=false;teste4:=false;
  custo:=inf;
  if cl[b1]=cl[b3] then
    begin
      teste1:=true;
      t:=1;
      custo:=c[ivec(a1,a3)]+c[ivec(b1,b3)]
                           -c[ivec(a1,b1)]-c[ivec(a3,b3)];
    end;
  if cl[b2]=cl[b4] then
   if custo>(c[ivec(a2,a4)]+c[ivec(b2,b4)]
                           -c[ivec(a2,b2)]-c[ivec(a4,b4)]) then
     begin
       teste2:=true;
       t:=2;
       custo:=c[ivec(a2,a4)]+c[ivec(b2,b4)]
                            -c[ivec(a2,b2)]-c[ivec(a4,b4)];
     end;
  if cl[b1]=cl[b4] then
    if custo>(c[ivec(a2,a4)]+c[ivec(a1,t1)]+c[ivec(b1,b4)]
          -c[ivec(a1,b1)]-c[ivec(a4,b4)]-c[ivec(a2,t1)]) then
      begin
        teste3:=true;
        t:=3;
        custo:=c[ivec(a2,a4)]+c[ivec(a1,t1)]+c[ivec(b1,b4)]
                  -c[ivec(a1,b1)]-c[ivec(a4,b4)]-c[ivec(a2,t1)];
      end;
  if cl[b2]=cl[b3] then
    if custo>(c[ivec(a2,a4)]+c[ivec(b2,b3)]+c[ivec(a3,t2)]
          -c[ivec(a2,b2)]-c[ivec(a3,b3)]-c[ivec(a4,t2)]) then
      begin
        teste4:=true;
        t:=4;
        custo:=(c[ivec(a2,a4)]+c[ivec(b2,b3)]+c[ivec(a3,t2)]
                  -c[ivec(a2,b2)]-c[ivec(a3,b3)]-c[ivec(a4,t2)])
      end;
```

```
  if not(teste1 or teste2 or teste3 or teste4) then
    begin
      custo:=c[ivec(a1,a3)]+c[ivec(b1,b3)]-c[ivec(a1,b1)]
                                          -c[ivec(a3,b3)];
      t:=1;
      if custo>(c[ivec(a2,a4)]+c[ivec(b2,b4)]-c[ivec(a2,b2)]
                                            -c[ivec(a4,b4)]) then
        begin
          t:=2;
          custo:=c[ivec(a2,a4)]+c[ivec(b2,b4)]-c[ivec(a2,b2)]
                                             -c[ivec(a4,b4)];
        end;
      if custo>(c[ivec(a2,a4)]+c[ivec(a1,t1)]+c[ivec(b1,b4)]
            -c[ivec(a1,b1)]-c[ivec(a4,b4)]-c[ivec(a2,t1)]) then
        begin
          t:=3;
          custo:=c[ivec(a2,a4)]+c[ivec(a1,t1)]+c[ivec(b1,b4)]
               -c[ivec(a1,b1)]-c[ivec(a4,b4)]-c[ivec(a2,t1)];
        end;
      if custo>(c[ivec(a2,a4)]+c[ivec(b2,b3)]+c[ivec(a3,t2)]
            -c[ivec(a2,b2)]-c[ivec(a3,b3)]-c[ivec(a4,t2)]) then
        begin
          t:=4;
          custo:=(c[ivec(a2,a4)]+c[ivec(b2,b3)]+c[ivec(a3,t2)]
                -c[ivec(a2,b2)]-c[ivec(a3,b3)]-c[ivec(a4,t2)])
        end;
    end;

end;  (*selectalt*)


procedure altcir;

(* este procedimento executa a alternativa seleccionada *)

begin
  l:=p;
    case t of

       1:begin
           pb1:=pa1^.prox;
           pb3:=pa3^.prox;
           l:=pa1^.ant;
           temp:=l^.ant;
           l^.ant:=l^.prox;
           l^.prox:=temp;
           a:=l;
           l:=l^.prox;
             repeat
               temp:=l^.ant;
               l^.ant:=l^.prox;
               l^.prox:=temp;
               temp:=l;
               l:=l^.prox;
             until l^.x=b3;
```

```
    l^.ant:=temp;
    pa1^.prox:=a;
    pa3^.prox:=pa1;
    pa1^.ant:=pa3;
    pa3^.prox:=pa1;
    pb1^.ant:=pb3;
    pb3^.prox:=pb1;
    pb1^.ant:=pb3;
    a:=pb1;
  end;

2:begin
    pb4:=pa4^.ant;
    pb2:=pa2^.ant;
    l:=pb4^.ant;
    temp:=l^.ant;
    l^.ant:=l^.prox;
    l^.prox:=temp;
    a:=l;
      repeat
        temp:=l^.ant;
        l^.ant:=l^.prox;
        l^.prox:=temp;
        temp:=l;
        l:=l^.prox;
      until l^.x=a2;
    l^.ant:=temp;
    pb4^.prox:=a;
    pb2^.prox:=pb4;
    pb4^.ant:=pb2;
    pa4^.ant:=pa2;
    pa2^.prox:=pa4;
    a:=pb4;
  end;

3:begin
    pb4:=pa4^.ant;
    pb1:=pa1^.prox;
    pt1:=pa2^.prox;
    pb1^.ant:=pb4;
    pb4^.prox:=pb1;
    pa1^.prox:=pt1;
    pt1^.ant:=pa1;
    pa2^.prox:=pa4;
    pa4^.ant:=pa2;
    a:=pa1;
  end;

4:begin
    pb2:=pa2^.ant;
    pb3:=pa3^.prox;
    pt2:=pa4^.prox;
    pb2^.prox:=pb3;
    pb3^.ant:=pb2;
```

```
            pa3^.prox:=pt2;
            pt2^.ant:=pa3;
            pa2^.ant:=pa4;
            pa4^.prox:=pa2;
            a:=pa4;
          end;
    end;(*case*)

end;(*altcir*)


PROCEDURE vizprox(
            n          :integer;
            var pesovp :real;
            pervp      :vecni);

(* este procedimento determina n circuitos Hamiltonianos pelo
   método do vizinho mais próximo e selecciona o de menor peso *)

var s,i,j,ind,vizprox  :integer;
    distmin,peso       :real;
    ciclo              :vecni;

begin
  pesovp:=inf*n;
  for s:=1 to n do
    begin
      for i:=1 to n do ciclo[i]:=0;
      peso:=0;
      ind:=s;
      for i:=1 to n-1 do
        begin
          distmin:=inf;
          ciclo[ind]:=ind;
          for j:=1 to n do
            if ciclo[j]=0 then
              if c[ivec(ind,j)]<distmin then
                begin
                  distmin:=c[ivec(ind,j)];
                  vizprox:=j
                end;
          peso:=peso+distmin;
          ciclo[ind]:=vizprox;
          ind:=vizprox;
        end;
      peso:=peso+c[ivec(s,ind)];
      if peso<pesovp then
                      begin
                        ind:=s;
                        for i:=1 to n do
                          begin
                            percvp[i]:=ind;
                            ind:=ciclo[ind];
                          end;
                      end;
    end;

end; (*vizprox*)
```

```
PROCEDURE mia(
            n              :integer;
            inf            :real;
            var c          :vec0mr;
            var perc       :vecni;
            var pesotm     :real);

(* este procedimento determina n circuitos Hamiltonianos pelo
   método da inserção mais afastada e selecciona o de menor peso *)

VAR  nter1,nter2,mafast,ind,proxind,i,j,s :integer;
     custins,maxdist,novocust,pesot       :real;
     ciclo                                :vecni;
     dist                                 :vecnr;

begin
  pesotm:=inf*n;
  for s:=1 to n do
    begin
        for i:=1 to n do ciclo[i]:=0;
        ciclo[s]:=s;
        for i:=1 to n do dist[i]:=c[ivec(i,s)];
        pesot:=inf;
        for i:=1 to n-1 do
          begin
            maxdist:=-inf;
            for j:=1 to n do
              if ciclo[j]=0 then
                if dist[j]>maxdist then
                  begin
                    maxdist:=dist[j];
                    mafast:=j
                  end;
            custins:=inf;
            ind:=s;
            for j:=1 to n do
              begin
                proxind:=ciclo[ind];
                novocust:=c[ivec(ind,mafast)]
                                        +c[ivec(mafast,proxind)]
                                        -c[ivec(ind,proxind)];
                if novocust<custins then
                  begin
                    custins:=novocust;
                    nter1:=ind;
                    nter2:=proxind
                  end;
                ind:=proxind;
              end;
            ciclo[mafast]:=nter2;
            ciclo[nter1]:=mafast;
            pesot:=pesot+custins;
            for j:=1 to n do
              if ciclo[j]=0 then
                if c[ivec(mafast,j)]<dist[j] then
                                  dist[j]:=C[ivec(mafast,j)]
          end;
```

```
        if pesotm>pesot then
                          begin
                            pesotm:=pesot;
                            ind:=s;
                            for i:=1 to n do
                              begin
                                perc[i]:=ind;
                                ind:=ciclo[ind]
                              end;
                          end;
    end;

end;(*mia*)

(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)
(*INSTRUCOES PRINCIPAIS*)

begin
  x1:=clock;
  ledados;
  x2:=clock;
  mia(n,inf,c,perc,pesotm);
  vizprox(n,pesovp,percvp);
  if pesovp<pesotm then
                     begin
                       pesotm:=pesovp;
                       clista(percvp)
                     end
                   else clista(perc);
  seleccl;
  if gm>2 then
            repeat
              selecabt;
              selecalt;
              pesotm:=pesotm+custo;
              altcir;
              seleccl;
           until gm=2;
  x3:=clock;

  (*output*)

  writeln('                    MÉTODO DE JONGENS E VOLGENANT');
  writeln;
  writeln;
  write('          PERCURSO OBTIDO:');
  custo:=0;
  i:=1;
  j:=trunc(n/20)+1;
  for t:=1 to j do
    begin
        repeat
          write(a^.x:3);
          custo:=custo+c[ivec(a^.x,a^.prox^.x)];
          a:=a^.prox;
          i:=i+1;
        until i=n+1;
    end;
```

```
  writeln;
  writeln('          O CUSTO DESTE PERCURSO É:  ',pesotm);
  writeln;
  writeln;
  writeln('          VERIFICAÇÃO DO CUSTO DO PERCURSO ',custo);
  writeln;
  writeln('     tempo de cpu sem io :','(x3-x2)/1000',' segundos');
  x2:=clock;
  writeln('     tempo de cpu com io :','(x2-x1)/1000',' segundos');
end.
```

EXEMPLO DE OUTPUT:

```
        MÉTODO DE JONGENS E VOLGENANT

 PERCURSO OBTIDO: 11  6  5  4  3  1  2 13 12 10  7  8  9
 O CUSTO DESTE PERCURSO É:   5.70340E+03

 VERIFICAÇÃO DO CUSTO DO PERCURSO  5.70340E+03

 tempo de cpu sem io :(x3-x2)/1000 segundos
 tempo de cpu com io :(x2-x1)/1000 segundos
```

A.3.2 - LISTAGEM DOS PROGRAMAS PARA A DETERMINAÇÃO DE MINORANTES

Os seis programas utilizados para determinar minorantes para o valor óptimo do PCVC são muito semelhantes. Assim, neste anexo apresenta-se apenas a listagem dos programas dos métodos M1E3 e M2E2. O primeiro programa exemplifica a primeira forma de determinação de minorantes (com subgradientes de dimensão n+1) e a terceira estratégia para o passo do método subgradiente. O segundo programa exemplifica a segunda forma de determinação de minorantes (com subgradientes de dimensão n+m) e a segunda estratégia para o passo do método subgradiente.

Para a utilização destes programas é necessário criar um ficheiro dados.dat já referido neste anexo. Para além disso o programa pede ao utilizador um majorante para o valor óptimo do problema.

Os exemplos de output que se seguem à listagem de cada programa referem-se ao ficheiro de input apresentado no início deste anexo.

A.3.2.1 - PROGRAMA DO MÉTODO M1E3

VARIÁVEIS DE INPUT

DADOS - ficheiro de input que contém o número de cidades, o número de classes, a classe a que pertence cada uma das cidades,um real superior a qualquer das distâncias e a matriz de distâncias;

MW - majorante do valor óptimo

N - número de cidades;

NUMCL - número de classes;

INF - real superior a qualquer das distâncias;

CL - vector que na posição i contém a classe a que pertence a cidade i;

CO - vector de distâncias.

VARIÁVEIS DE OUTPUT

PESOTO - valor melhor minorante obtido.

TARE1OP,TARE2OP - vectores que contêm os nodos terminais das arestas que constituem a solução da melhor relaxação.

LISTAGEM DO PROGRAMA:

```
PROGRAM M1E3 (input,output,dados);

(* ESTE PROGRAMA DETERMINA UM MINORANTE PARA O PCVC PELO MÉTODO
   M1E3 *)

LABEL 100;

CONST n1=151;   (*n de cidades+1*)
      m1=(sqr(n1-1)-n1+1)div 2;  (*num de arestas (sqr(n)-n)/2 *)

TYPE  vecmr=array[1..m1] of real;
      vecnr=array[1..n1] of real;
      vecni=array[1..n1] of integer;
      vecmi=array[1..m1] of integer;

VAR   co,c                                                  :vecmr;
      mw,p,inf,x1,x2,x3,x4,pk,pesot,pesoto,e1,e,g2          :real;
      k1,a1,b1,n,numcl,nanaol,i,j,l,k                       :integer;
      tare1,tare2,tare1op,tare2op,cl                        :vecni;
      g,u,uo                                                :vecnr;
      convergiu                                             :boolean;
      dados                                                 :text;


FUNCTION ivec (I,J: integer):integer;

(* esta função recebe os índices de uma matriz (triângular
   superior I<J) e transforma-os no índice de um vector *)

VAR t:integer;

begin
  if i>j then
           begin
             t:=i;
             i:=j;
             j:=t
           end;
  ivec:=((i-1)*n-((1+i)*i)div 2 )+j
end; (*ivec*)
```

```
PROCEDURE ledados;

(* lê os dados de input *)

var i,j:integer;

begin
  write('qual o majorante ? ');
  readln(mw);
  reset(dados);
  read(dados,n);
  read(dados,numcl);
  for i:=1 to n do read(dados,cl[i]);
  read(dados,inf);
  for i:=1 to (sqr(n)-n)div 2 do read(dados,co[i]);

end(*ledados*);


PROCEDURE arvore1_prim (
                  inf              :real;
                  c                :vecmr;
                  var tare1,tare2  :vecni;
                  var pesot        :real);

VAR i,j,l          :integer;
    menor1,menor2  :real;

  PROCEDURE prim (
                n                :integer;
                inf              :real;
                var c            :vecmr;
                var tare1,tare2  :vecni;
                var pesot        :real);



  VAR u,k,tcount,i,j,l                :integer;
      min                             :real;
      viz                             :vecni;
      dist                            :vecnr;


   begin
     viz[2]:=0;
     for i:=3 to n do
       begin
         viz[i]:=2;
         dist[i]:=c[ivec(2,i)];
       end;
     tcount:=0;
     pesot:=0;
```

```
  while (tcount<n-2) do
    begin
      min:=inf;
      for k:=3 to n do
        if viz[k]<>0 then
          if dist[k]<min then
            begin
              u:=k;
              min:=dist[k]
            end;
      tcount:=tcount+1;
      tare1[tcount]:=viz[u];
      tare2[tcount]:=u;
      viz[u]:=0;
      for k:=3 to n do
      (*actualizacao do vizinho*)
        if viz[k]<>0 then
          if c[ivec(k,viz[k])]>c[ivec(u,k)] then
            begin
              dist[k]:=c[ivEC(k,u)];
              viz[k]:=u
            end
    end(*while*);
end(*prim*);



begin
  prim(n,inf,c,tare1,tare2,pesot);
  (*2 arestas menores incidentes no nodo 1*)
  menor1:=inf;menor2:=inf;tare1[n-1]:=1;tare2[n-1]:=0;
  tare1[n]:=1;tare2[n]:=0;
  for i:=2 to n do
     if c[ivec(1,i)]<menor1 then
       begin
         menor1:=c[ivec(1,i)];
         tare2[n-1]:=i
       end;
  for i:=2 to n do
     if (c[ivec(1,i)]<menor2) and (i<>tare2[n-1]) then
        begin
          menor2:=C[ivec(1,I)];
          tare2[n]:=i
        end;


  pesot:=0;
  for i:=1 to n do pesot:=pesot+co[ivec(tare1[i],tare2[i])];
end;(*arvore1_prim*)
```

```
PROCEDURE gradenorma (
                    var g2              :real;
                    var nanaol          :integer;
                    var g               :vecnr);

(* este procedimento calcula um subgradiente e o quadrado
                                                  da sua norma *)

var j :integer;

begin
  for j:=1 to n do g[j]:=-2;
  for j:=1 to n do
    begin
      g[tare1[j]]:=g[tare1[j]]+1;
      g[tare2[j]]:=g[tare2[j]]+1
    end;
  nanaol:=-numcl;
  for j:=1 to n do
    if cl[tare1[j]]<>cl[tare2[j]] then nanaol:=nanaol+1;
  g[n+1]:=nanaol;
  g2:=0;
  for j:=1 to n+1 do g2:=g2+g[j]*g[j];

end; (*gradenorma*)



(*INSTRUCOES DO PROGRAMA*)
(*INSTRUCOES DO PROGRAMA*)
(*INSTRUCOES DO PROGRAMA*)



begin
  x3:=clock;
  ledados;
  x1:=clock;
  pesot:=0;
  for l:=1 to n+1 do u[l]:=0;
  for l:=1 to n+1 do uo[l]:=0;
  arvore1_prim (inf, co,tare1,tare2,pesot);
  pesoto:=pesot;
  for l:=1 to n do
    begin
      tare1op[l]:=tare1[l];
      tare2op[l]:=tare2[l]
    end;
  convergiu:=false;
  k:=1;
  gradenorma(g2,nanaol,g);
  e:=mw-pesoto;
  a1:=n div 2;
  b1:=0;
  p:=1;
  if (g2=0) then convergiu:=true;
  while not (convergiu) and (k<400) do
```

```
(* determinação do auxiliar do passo do método subgradiente *)
  begin
    if b1>=a1 then if a1>=5 then
                              begin
                                a1:=a1 div 2;
                                b1:=1;
                                p:=p/2;
                              end
                            else
                              begin
                                a1:=5;
                                b1:=1;
                                p:=p/2;
                              end
              else b1:=b1+1;

    pk:=p*(mw-pesot)/g2;
    for l:=1 to n+1 do u[l]:=u[l]+pk*g[l];


    for i:=1 to n do for j:=i+1 to n do
    if cl[i]=cl[j] then c[ivec(i,j)]:=co[ivec(i,j)]+u[i]+u[j]
                   else c[ivec(i,j)]:=co[ivec(i,j)]+u[i]+u[j]+u[n+1];

    arvore1_prim (inf, c,tare1,tare2,pesot);
    gradenorma (g2,nanaol,g);
    for i:=1 to n+1 do pesot:=pesot+u[i]*g[i];
    k:=k+1;
    if (mw-pesot)<0 then
      writeln('ERRO minorante obtido é superior ao majorante');
    (* actualizacao do melhor minorante conseguido *)
    if (mw-pesot)<e then
      begin
        e:=abs(mw-pesot);
        pesoto:=pesot;
        k1:=k;
        for l:=1 to n+1 do uo[l]:=u[l];
        for l:=1 to n do
          begin
            tare1op[l]:=tare1[l];
            tare2op[l]:=tare2[l];
          end;
      end;
    if g2=0 then convergiu:=true;
    if k-k1=50 then goto 100;
  end(*while*);
100:e1:=(e/pesoto)*100;
E:=(E/MW)*100;
x2:=clock;
(*output*)
writeln;
writeln;
writeln;
writeln('       RESULTADOS DA APLICAÇÃO DE M1E3');
writeln;
writeln;
```

```
  if g2<>0 then
   begin
     writeln('       o num de iterações foi ',k);
     write('         o num da iteração em que foi encontrado o ');
     writeln('melhor valor ',k1);
     write('         o erro cometido e majorado por ',e:3);
     writeln( '% do majorante');
     write('         o erro cometido e majorado por ',e1:3);
     writeln('% do minorante');
     writeln;
     writeln('           o melhor minorante encontrado é ',pesoto);
   end
          else
   begin
     writeln('    foi encontrado um percurso óptimo formado por:');
     writeln;
     for l:=1 to n do writeln('        ',tare1op[l],tare2op[l]);
     writeln;
     writeln('       o respectivo custo é ',pesoto);
     writeln('       o num de iterações foi ',k);
   end;
  writeln;
  writeln;
  writeln('       tempo de cpu sem io e : ',(x2-x1)/1000,' seg');
  x4:=clock;
  writeln('       tempo de cpu com io e : ',(x4-x3)/1000,' seg');

end.
```

EXEMPLO DE OUTPUT:

```
qual o majorante ? 5127.23

RESULTADOS DA APLICAÇÃO DE M1E3


foi encontrado um percurso óptimo formado por:

         2        12
        12        13
        13        11
        11         7
         7        10
        10         8
         8         9
         9         6
         6         5
         5         4
         4         3
         1         2
         1         3

o respectivo custo é  4.99189E+03
o num de iterações foi         16


tempo de cpu sem io e :  8.20000E-01 seg
tempo de cpu com io e :  1.03000E+00 seg
```

A.3.2.2 - PROGRAMA DO MÉTODO M2E2

VARIÁVEIS DE INPUT

DADOS - ficheiro de input que contém o número de cidades, o número de classes, a classe a que pertence cada uma das cidades,um real superior a qualquer das distâncias e a matriz de distâncias;

MW - majorante do valor óptimo

N - número de cidades;

NUMCL - número de classes;

INF - real superior a qualquer das distâncias;

CL - vector que na posição i contém a classe a que pertence a cidade i;

CO - vector de distâncias.

VARIÁVEIS DE OUTPUT

PESOTO - valor melhor minorante obtido.

TARE1OP,TARE2OP - vectores que contêm os nodos terminais das arestas que constituem a solução da melhor relaxação.

LISTAGEM DO PROGRAMA:

```
PROGRAM M2E2 (input,output,dados);

(* ESTE PROGRAMA DETERMINA UM MINORANTE PARA O PCVC PELO MÉTODO
   M2E2 *)

LABEL 100;

CONST n1=150;   (*n de cidades*)
      m1=(sqr(n1-1)-n1+1)div 2;  (*num de arestas (sqr(n)-n)/2 *)
      numcl1=30; (*numero de classes*)
      nm=n1+numcl1; (*para dimensionar o subgradiente*)

TYPE  vecmr=array[1..m1] of real;
      vecnr=array[1..n1] of real;
      vecni=array[1..n1] of integer;
      vecmi=array[1..m1] of integer;
      vecnumcli=array[1..numcl1] of integer;
      vecnmr=array[1..nm] of real;
```

```
VAR   co,c                                              :vecmr;
      mw,p,inf,x1,x2,x3,x4,pesot,pesoto,e1,
      e,g2,a,pk                                         :real;
      k1,a1,b1,n,numcl,nanaol,i,j,l,k,b,z               :integer;
      tare1,tare2,tare1op,tare2op,cl                    :vecni;
      nncl                                              :vecnumcli;
      g,u,uo                                            :vecnmr;
      convergiu                                         :boolean;
      dados                                             :text;


FUNCTION ivec (I,J: integer):integer;

(* esta função recebe os índices de uma matriz (triângular
   superior I<J) e transforma-os no índice de um vector *)

VAR t:integer;

begin
  if i>j then
           begin
             t:=i;
             i:=j;
             j:=t
           end;
  ivec:=((i-1)*n-((1+i)*i)div 2 )+j
end; (*ivec*)


PROCEDURE ledados;

(* lê os dados de input *)

var i,j:integer;

begin
  write('qual o majorante ? ');
  readln(mw);
  reset(dados);
  read(dados,n);
  read(dados,numcl);
  for i:=1 to n do read(dados,cl[i]);
  read(dados,inf);
  for i:=1 to (sqr(n)-n)div 2 do read(dados,co[i]);

end(*ledados*);


PROCEDURE arvore1_prim (
                  inf              :real;
                  c                :vecmr;
                  var tare1,tare2  :vecni;
                  var pesot        :real);

VAR i,j,l         :integer;
    menor1,menor2 :real;
```

```
PROCEDURE prim (
                n                 :integer;
                inf               :real;
                var c             :vecmr;
                var tare1,tare2   :vecni;
                var pesot         :real);


VAR u,k,tcount,i,j,l                    :integer;
    min                                 :real;
    viz                                 :vecni;
    dist                                :vecnr;


 begin
   viz[2]:=0;
   for i:=3 to n do
     begin
       viz[i]:=2;
       dist[i]:=c[ivec(2,i)];
     end;
   tcount:=0;
   pesot:=0;

   while (tcount<n-2) do
     begin
       min:=inf;
       for k:=3 to n do
         if viz[k]<>0 then
           if dist[k]<min then
             begin
               u:=k;
               min:=dist[k]
             end;
       tcount:=tcount+1;
       tare1[tcount]:=viz[u];
       tare2[tcount]:=u;
       viz[u]:=0;
       for k:=3 to n do
       (*actualizacao do vizinho*)
         if viz[k]<>0 then
           if c[ivec(k,viz[k])]>c[ivec(u,k)] then
             begin
               dist[k]:=c[ivEC(k,u)];
               viz[k]:=u
             end
     end(*while*);
 end(*prim*);
```

```
  begin
    prim(n,inf,c,tare1,tare2,pesot);
    (*2 arestas menores incidentes no nodo 1*)
    menor1:=inf;menor2:=inf;tare1[n-1]:=1;tare2[n-1]:=0;
    tare1[n]:=1;tare2[n]:=0;
    for i:=2 to n do
       if c[ivec(1,i)]<menor1 then
         begin
           menor1:=c[ivec(1,i)];
           tare2[n-1]:=i
         end;
    for i:=2 to n do
       if (c[ivec(1,i)]<menor2) and (i<>tare2[n-1]) then
          begin
            menor2:=C[ivec(1,I)];
            tare2[n]:=i
          end;


    pesot:=0;
    for i:=1 to n do pesot:=pesot+co[ivec(tare1[i],tare2[i])];
  end;(*arvore1_prim*)



PROCEDURE gradenorma (
                  var g2            :real;
                  var g             :vecnmr;
                  nncl              :vecnumcli);

(* este procedimento calcula um subgradiente e o quadrado
                                                  da sua norma *)

var j :integer;

begin
  for j:=1 to n+numcl do g[j]:=-2;
  for j:=1 to n do
    begin
      g[tare1[j]]:=g[tare1[j]]+1;
      g[tare2[j]]:=g[tare2[j]]+1
    end;
  for j:=1 to n do
    if cl[tare1[j]]<>cl[tare2[j]] then
      begin
        g[n+cl[tare1[j]]]:=g[n+cl[tare1[j]]]+1;
        g[n+cl[tare2[j]]]:=g[n+cl[tare2[j]]]+1
      end;
  g2:=0;
  for j:=1 to n+numcl do g2:=g2+g[j]*g[j];

end; (*gradenorma*)
```

```
(*INSTRUCOES DO PROGRAMA*)
(*INSTRUCOES DO PROGRAMA*)
(*INSTRUCOES DO PROGRAMA*)



begin
  x3:=clock;
  ledados;
  x1:=clock;
  pesot:=0;
  for i:=1 to numcl do nncl[i]:=0;
  for i:=1 to n do nncl[cl[i]]:=nncl[cl[i]]+1;
  for l:=1 to n+numcl do u[l]:=0;
  for l:=1 to n+numcl do uo[l]:=0;
  arvore1_prim (inf, co,tare1,tare2,pesot);
  pesoto:=pesot;
  for l:=1 to n do
    begin
      tare1op[l]:=tare1[l];
      tare2op[l]:=tare2[l]
    end;
  convergiu:=false;
  k:=1;
  gradenorma(g2,g,nncl);
  e:=mw-pesoto;
  p:=2;
  b:=0;
  if (g2=0) then convergiu:=true;
  while not (convergiu) and (k<400) do

  (* determinação do auxiliar do passo do método subgradiente *)
    begin
      if b=4 then
                begin
                  p:=p/2;
                  b:=0
                end
              else b:=b+1;

      pk:=p*(mw-pesot)/g2;
      for l:=1 to n+numcl do u[l]:=u[l]+pk*g[l];


      for i:=1 to n do for j:=i+1 to n do
      if cl[i]=cl[j] then c[ivec(i,j)]:=co[ivec(i,j)]+u[i]+u[j]
                     else c[ivec(i,j)]:=co[ivec(i,j)]+u[i]+u[j]
                                        +u[n+cl[i]]+u[n+cl[j]];

      arvore1_prim (inf, c,tare1,tare2,pesot);
      gradenorma (g2,g,nncl);
      for i:=1 to n+numcl do pesot:=pesot+u[i]*g[i];
      k:=k+1;
      if (mw-pesot)<0 then
        writeln('ERRO minorante obtido é superior ao majorante');
```

```
    (* actualizacao do melhor minorante conseguido *)
    if (mw-pesot)<e then
      begin
        b:=0;
        e:=abs(mw-pesot);
        pesoto:=pesot;
        k1:=k;
        for l:=1 to n+numcl do uo[l]:=u[l];
        for l:=1 to n do
          begin
            tare1op[l]:=tare1[l];
            tare2op[l]:=tare2[l];
          end;
      end;
    if g2=0 then convergiu:=true;
    if k-k1=50 then goto 100;
  end(*while*);

100:e1:=(e/pesoto)*100;
E:=(E/MW)*100;
x2:=clock;
(*output*)
writeln;
writeln;
writeln;
writeln('       RESULTADOS DA APLICAÇÃO DE M2E2');
writeln;
writeln;
if g2<>0 then
 begin
   writeln('       o num de iterações foi ',k);
   write('         o num da iteração em que foi encontrado o ');
   writeln('melhor valor ',k1);
   write('         o erro cometido e majorado por ',e:3);
   writeln( '% do majorante');
   write('         o erro cometido e majorado por ',e1:3);
   writeln('% do minorante');
   writeln;
   writeln('           o melhor minorante encontrado é ',pesoto);
 end
        else
 begin
   writeln('    foi encontrado um percurso óptimo formado por:');
   writeln;
   for l:=1 to n do writeln('        ',tare1op[l],tare2op[l]);
   writeln;
   writeln('       o respectivo custo é ',pesoto);
   writeln('       o num de iterações foi ',k);
 end;
writeln;
writeln;
writeln('       tempo de cpu sem io e : ',(x2-x1)/1000,' seg');
x4:=clock;
writeln('       tempo de cpu com io e : ',(x4-x3)/1000,' seg');

end.
```

EXEMPLO DE OUTPUT:

```
qual o majorante ? 5127.23

RESULTADOS DA APLICAÇÃO DE M2E2

foi encontrado um percurso óptimo formado por:

          2         12
         12         13
         13         11
         11          7
          7         10
         10          8
          8          9
          9          6
          6          5
          5          4
          4          3
          1          2
          1          3

o respectivo custo é  4.99189E+03
o num de iterações foi          19


tempo de cpu sem io e :  9.50000E-01 seg
tempo de cpu com io e :  1.19000E+00 seg
```

REFERÊNCIAS

Among S. (1954), The Relaxation Method for Linear Inequalities, Canadian Journal of Mathematics 6, pp 382-92.

Balas E., Christofides (1981), A Restricted Lagrangean Approach to the Traveling Salesman Problem, Mathematical Programming 21, pp 19-46.

Balas E., P.Toth (1985), Branch and Bound Methods, in the Traveling Salesman Problem -A guided tour of Combinatorial Optimization. Lawer E.L., J. K. Lenstra , A.H.G. Rinnoy Kan, D.B. Shmoys. (eds) John Wiley and Sons Ltd.

Bárcia P. (1985), Métodos de Reforço do Dual em Programação Linear Inteira, tese de doutoramento, Instituto Superior Técnico.

Bazaraa M.S., H.D. Sherali (1981), On the Choice of Step Size in Subgradient Optimization, European Journal of Operations Research 7, pp 380-388.

Bell D.E., Shapiro J.F. (1977), A Convergent Duality Theory for Integer Programming, Operations Research vol 25, nº 3, pp 419-434.

Chisman J. (1975), The Clustered Traveling Salesman Problem, Computers and Operations Reseach, vol 2, pp 115--119.

Christofides N. (1979), The Travelling Salesman Problem, in Combinatorial Optimization, Christofides N., A. Mingozzi, P. Toth, C.Sandi (eds) John Wiley and Sons.

Crowder H., M.W. Padberg (1980), Solving Large Scale Traveling Salesman Problems to Optimality, Management Science 26, pp 495-509.

Dantzig G.B., D.R. Fulkerson, S.M. Johnson (1954), Solution of a Large Scale Traveling Salesman Problem, Operations Research 2, pp 393-410.

Garey M.R., D.S. Johnson (1979), Computers and Intractability--A Guide to the Theory of NP-Completeness, M.H. Freeman and Company.

Garfinkel R.S. (1985), Motivation and Modeling, in The Traveling Salesman Problem -A Guided Tour of Combinatorial Optimization Lawer E.L., J.K. Lenstra, A.H.G. Rinnooy e D.B. Shmoys (eds) John Wiley and Sons Ltd.

Geoffrion A. (1974), Lagrangean Relaxation for Integer Programming, Mathematical Programming Study 2, pp 82-114.

Golden B., L. Bodin, T. Doyle, W. Stewart Jr (1980), Approximate Traveling Salesman Algorithms, Operations Research vol 18, nº 3, part II.

Grotschel M. (1980), On the Symmetric Traveling Salesman Problem: solution of a 120-city problem, Mathematical Programming Study 12, pp 61-77.

Grotschel M.,M.W. Padberg (1985),Polyedral Theory, in The Traveling Salesman Problem -A Guided Tour of Combinatorial Optimization Lawer E.L., J.K. Lenstra, A.H.G. Rinnooy e D.B. Shmoys John Wiley and Sons Ltd.

Held M., R. Karp (1962), A Dynamic Programming Approach to Sequencing Problems, Journal of the Society for Industrial and Applied Mathematics 10, pp 196--210.

Held M., R. Karp (1970) The Travelling Salesman Problem and Minimum Spanning Trees, Operations Research 18, 1138--1162.

Held M., R. Karp (1971) The Travelling Salesman Problem and Minimum Spanning Trees II, Mathematical Programming, pp 6-25.

Held M., P. Wolfe, H.P. Crowder (1974), Validation of Subgradient Optimization, Mathematical Programming 6, pp 62--88.

Hoffman A.J., P. Wolfe (1985), History, in The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization Lawer E.L., J.K. Lenstra, A.H.G. Rinnooy e D.B. Shmoys John Wiley and Sons Ltd.

Jongens K., T. Volgenant (1985), The Symmetric Clustered Traveling Problem,European Journal of Operational Research 19, pp 68-75.

Kruskal J.B. (1956), On the Shortest Spanning Subtree of a Graph and Traveling Salesman Problem, Procedings American Mathematical Society 2, pp 48-50.

Lin S. (1965), Computer Solution of the Traveling Salesman Problem, Bell System Technical Journal 44, pp 2245-2269.

Little J.D.C., K.G. Murty, D.W. Sweeney, C. Karel (1963), An Algorithm for the Traveling Salesman Problem, Operations Research 11, pp 972-989.

Lokin F. (1978), Procedures for Travelling Salesman Problems with Addicional Constrains, European Journal of Operational Research 3, pp 135-141.

Magnanti T.L., J.F. Shapiro, M.H. Wagner (1976), Generalized Linear Programming Solves the Dual, Managment Science vol 22, nº 11, pp 1195-1203.

Motzkin T., I.J. Schoenberg (1954),The Relaxation Method for Linear Inequalities, Canadian Journal of Mathematics 6, pp 393-404.

Padberg M., G. Rinaldi (1986), Optimization of 532-City Symmetric Travelling Salesman Problem, Journée du 20-éme Aniversaire du Groupe Combinatoire, AFCET-3,4,5 Dec.1986.

Prim R.C. (1957), Shortest Connection Networks and Some Generalizations , Bell System Technical Journal 36, pp 1389-1401.

Rozenkrantz D., R. Stearns, P. Lewis (1974), Approximate Algorithms for the Traveling Salesman Problem, Proceeding of the 15 th Annual IEEE Symposium of Switching and Automata Theory, pp 33-42.

Shapiro J. (1979), Mathematical Programming: Structures and Alghoritms, John Wiley and Sons.

Syslo M.M., N. Deo, J.S. Kowalik (1983), Discrete Optimization Algorithms with Pascal Programs, Prentice-Hall, Inc. Englewood Cliffs.

Távora J. (1985), Optimização do Percurso da Ferramenta de Corte de Tecido, comunicação apresentada no Terceiro Encontro Português de Inteligência Artificial, Braga.

Wagner M.M. (1969), Principles of Operations Research with Applications to Managerial Decisions, Prentice-Hall.